# Fast Models

Version 11.23

# Fixed Virtual Platforms (FVP) Reference Guide

**Non-Confidential**

Copyright © 2017–2023 Arm Limited (or its affiliates).
All rights reserved.

**Issue 00**
100966_1123_00_en

# Fast Models

## Fixed Virtual Platforms (FVP) Reference Guide

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.

## Release Information

### Document history

| Issue | Date | Confidentiality | Change |
|---|---|---|---|
| 1100-00 | 31 May 2017 | Non-Confidential | Update for v11.0. Document numbering scheme has changed. |
| 1101-00 | 31 August 2017 | Non-Confidential | Update for v11.1. |
| 1102-00 | 17 November 2017 | Non-Confidential | Update for v11.2. |
| 1103-00 | 23 February 2018 | Non-Confidential | Update for v11.3. |
| 1104-00 | 22 June 2018 | Non-Confidential | Update for v11.4. |
| 1104-01 | 17 August 2018 | Non-Confidential | Update for v11.4.2. |
| 1105-00 | 23 November 2018 | Non-Confidential | Update for v11.5. |
| 1106-00 | 26 February 2019 | Non-Confidential | Update for v11.6. |
| 1107-00 | 17 May 2019 | Non-Confidential | Update for v11.7. |
| 1108-00 | 5 September 2019 | Non-Confidential | Update for v11.8. |
| 1108-01 | 3 October 2019 | Non-Confidential | Update for v11.8.1. |
| 1109-00 | 28 November 2019 | Non-Confidential | Update for v11.9. |
| 1110-00 | 12 March 2020 | Non-Confidential | Update for v11.10. |

| Issue | Date | Confidentiality | Change |
|---|---|---|---|
| 1111-00 | 9 June 2020 | Non-Confidential | Update for v11.11. |
| 1112-00 | 22 September 2020 | Non-Confidential | Update for v11.12. |
| 1113-00 | 9 December 2020 | Non-Confidential | Update for v11.13. |
| 1114-00 | 17 March 2021 | Non-Confidential | Update for v11.14. |
| 1114-01 | 14 April 2021 | Non-Confidential | Update for FVP_Base_AEMv8R. |
| 1115-00 | 29 June 2021 | Non-Confidential | Update for v11.15. |
| 1116-00 | 6 October 2021 | Non-Confidential | Update for v11.16. |
| 1117-00 | 16 February 2022 | Non-Confidential | Update for v11.17. |
| 1118-00 | 15 June 2022 | Non-Confidential | Update for v11.18. |
| 1118-01 | 26 August 2022 | Non-Confidential | Update for Arm Corstone SSE-310 FVP. |
| 1119-00 | 14 September 2022 | Non-Confidential | Update for v11.19. |
| 1120-00 | 7 December 2022 | Non-Confidential | Update for v11.20. |
| 1121-00 | 22 March 2023 | Non-Confidential | Update for v11.21. |
| 1122-00 | 14 June 2023 | Non-Confidential | Update for v11.22. |
| 1123-00 | 13 September 2023 | Non-Confidential | Update for v11.23. |

## Proprietary Notice

This document is protected by copyright and other related rights and the practice or implementation of the information contained in this document may be protected by one or more

patents or pending patent applications. No part of this document may be reproduced in any form by any means without the express prior written permission of Arm. No license, express or implied, by estoppel or otherwise to any intellectual property rights is granted by this document unless specifically stated.

Your access to the information in this document is conditional upon your acceptance that you will not use or permit others to use the information for the purposes of determining whether implementations infringe any third party patents.

THIS DOCUMENT IS PROVIDED "AS IS". ARM PROVIDES NO REPRESENTATIONS AND NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY, SATISFACTORY QUALITY, NON-INFRINGEMENT OR FITNESS FOR A PARTICULAR PURPOSE WITH RESPECT TO THE DOCUMENT. For the avoidance of doubt, Arm makes no representation with respect to, and has undertaken no analysis to identify or understand the scope and content of, patents, copyrights, trade secrets, or other rights.

This document may include technical inaccuracies or typographical errors.

TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL ARM BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF ARM HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

This document consists solely of commercial items. You shall be responsible for ensuring that any use, duplication or disclosure of this document complies fully with any relevant export laws and regulations to assure that this document or any portion thereof is not exported, directly or indirectly, in violation of such export laws. Use of the word "partner" in reference to Arm's customers is not intended to create or refer to any partnership relationship with any other company. Arm may make changes to this document at any time and without notice.

This document may be translated into other languages for convenience, and you agree that if there is any conflict between the English version of this document and any translation, the terms of the English version of the Agreement shall prevail.

The Arm corporate logo and words marked with ® or ™ are registered trademarks or trademarks of Arm Limited (or its affiliates) in the US and/or elsewhere. All rights reserved. Other brands and names mentioned in this document may be the trademarks of their respective owners. Please follow Arm's trademark usage guidelines at https://www.arm.com/company/policies/trademarks.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

(LES-PRE-20349|version 21.0)

## Confidentiality Status

This document is Non-Confidential. The right to use, copy and disclose this document may be subject to license restrictions in accordance with the terms of the agreement entered into by Arm and the party that Arm delivered this document to.

Unrestricted Access is an Arm internal classification.

## Product Status

The information in this document is Final, that is for a developed product.

## Feedback

Arm welcomes feedback on this product and its documentation. To provide feedback on the product, create a ticket on https://support.developer.arm.com.

To provide feedback on the document, fill the following survey: https://developer.arm.com/documentation-feedback-survey.

## Inclusive language commitment

Arm values inclusive communities. Arm recognizes that we and our industry have used language that can be offensive. Arm strives to lead the industry and create change.

This document includes language that can be offensive. We will replace this language in a future issue of this document.

To report offensive language in this document, email terms@arm.com.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

# Contents

**1. Introduction....13**
1.1 Conventions....13
1.2 Other information....14
1.3 Useful resources....14

**2. Introduction to FVPs....16**
2.1 Types of FVP....16

**3. Getting Started with Fixed Virtual Platforms....18**
3.1 Requirements for FVPs....18
3.2 Contents of the FVP Standard Library package....19
3.3 FVP command-line options....20
3.4 Loading and running an application on an FVP....25
3.5 Configuring the model....26
3.6 FVP debug....27
3.7 Using the CLCD window....28
3.8 Ethernet with VE FVPs....31
3.9 Using a terminal with a system model....33
3.10 Virtio P9 device component....36

**4. Arm® CoreLink™ SGI-575 reference design FVP....38**
4.1 About the SGI-575 FVP....38
4.2 SGI-575 FVP peripherals....38
4.2.1 Memory map for SGI-575 FVP SoC peripherals....39
4.2.2 Memory map for SGI-575 FVP board peripherals....40
4.2.3 Interrupt maps for SGI-575 FVP....40

**5. Arm® Corstone™ SSE-300 FVP....42**
5.1 Memory map overview for Corstone™ SSE-300....42
5.2 Corstone™ SSE-300 FVP peripherals....44
5.2.1 Corstone™ SSE-300 non-secure expansion peripherals memory map....45
5.2.2 Corstone™ SSE-300 secure expansion peripherals memory map....46
5.2.3 Corstone™ SSE-300 expansion peripheral interrupt map....48

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

5.3 Corstone™ SSE-300 peripheral protection controller expansion map....50
5.4 Corstone™ SSE-300 memory components....50

**6. Arm® Corstone™ SSE-310 FVP....52**
6.1 Corstone™ SSE-310 FVP memory map overview....52
6.2 Corstone™ SSE-310 FVP peripherals....54
6.2.1 Corstone™ SSE-310 FVP subsystem peripherals non-secure memory map....55
6.2.2 Corstone™ SSE-310 FVP subsystem peripherals secure memory map....56
6.2.3 Corstone™ SSE-310 FVP non-secure expansion peripherals memory map....57
6.2.4 Corstone™ SSE-310 FVP secure expansion peripherals memory map....59
6.2.5 Corstone SSE-310 FVP expansion peripheral interrupt map....60
6.3 Corstone™ SSE-310 FVP peripheral protection controller expansion map....62
6.4 Corstone™ SSE-310 FVP memory components....62

**7. Arm® Corstone™-1000 FVP....64**
7.1 Corstone™-1000 FVP modeled components....64
7.2 Run the Corstone™-1000 FVP with the software package....65
7.3 Corstone™-1000 board peripherals memory and interrupt map....66
7.4 Networking on Corstone™-1000 FVP....66

**8. Juno FVP3 model....67**
8.1 Unimplemented features and model limitations....67
8.2 Running the Juno FVP3 model....68
8.2.1 Run the Juno FVP3 model in Arm DS....68
8.2.2 Run the Juno FVP3 model standalone....68
8.2.3 Run the create_afs_image.py script....69
8.3 Jump-start the application clusters on bare metal....70
8.4 Basic setup for the TZC-400 model on bare metal....70

**9. Arm® Neoverse™ N1 edge and Neoverse™ E1 edge reference design FVPs....72**
9.1 About the RD-N1-E1 FVPs....72
9.2 Block diagrams for RD-N1-E1-edge....73
9.3 RD-N1-E1 FVP peripherals....74
9.3.1 Memory map for RD-N1-E1-edge FVP SoC peripherals....75
9.3.2 Memory map for RD-N1-E1-edge FVP board peripherals....76
9.3.3 Interrupt maps for RD-N1-E1-edge FVP....77

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**10. Arm® Neoverse™ N2 reference design FVP....78**
10.1 About the RD-N2 FVP....78
10.2 RD-N2 FVP peripherals....79
10.2.1 Memory map for RD-N2 FVP SoC peripherals....79
10.2.2 Memory map for RD-N2 FVP board peripherals....80
10.2.3 Interrupt maps for RD-N2 FVP....81

**11. Arm® Neoverse™ V1 reference design FVP....83**
11.1 About the RD-V1 FVP....83
11.2 RD-V1 FVP peripherals....84
11.2.1 Memory map for RD-V1 FVP SoC peripherals....84
11.2.2 Memory map for RD-V1 FVP board peripherals....85
11.2.3 Interrupt maps for RD-V1 FVP....86

**12. Configurable PCIe hierarchy....88**
12.1 JSON format for the hierarchy....88
12.2 Common PCIe endpoint parameters....89
12.3 AHCI controller parameters....91
12.4 Host bridge parameters....93
12.5 SMMU test engine parameters....94
12.6 Root port parameters....96
12.7 Switch parameters....97
12.8 Root bridge parameters....97
12.9 Exerciser....98
12.10 Example hierarchy for a single ECAM configuration....100
12.11 Example hierarchy for two ECAM configuration....101

**13. Base Platform FVPs....102**
13.1 FVP_Base_AEMv8R....102
13.2 FVP_Base_AEMvA-AEMvA....121
13.3 FVP_Base_Cortex-A32x1....143
13.4 FVP_Base_Cortex-A32x2....161
13.5 FVP_Base_Cortex-A32x4....180
13.6 FVP_Base_Cortex-A34x1....200
13.7 FVP_Base_Cortex-A34x2....219
13.8 FVP_Base_Cortex-A34x4....238
13.9 FVP_Base_Cortex-A35x1....257

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

13.10 FVP_Base_Cortex-A35x2 ..... 276
13.11 FVP_Base_Cortex-A35x4 ..... 295
13.12 FVP_Base_Cortex-A510 ..... 314
13.13 FVP_Base_Cortex-A510+Cortex-A710 ..... 337
13.14 FVP_Base_Cortex-A510x4+Cortex-A710x4 ..... 356
13.15 FVP_Base_Cortex-A520 ..... 377
13.16 FVP_Base_Cortex-A53x1 ..... 401
13.17 FVP_Base_Cortex-A53x2 ..... 420
13.18 FVP_Base_Cortex-A53x4 ..... 439
13.19 FVP_Base_Cortex-A55 ..... 458
13.20 FVP_Base_Cortex-A55+Cortex-A76 ..... 480
13.21 FVP_Base_Cortex-A55x1+Cortex-A75x1 ..... 499
13.22 FVP_Base_Cortex-A55x2+Cortex-A75x2 ..... 518
13.23 FVP_Base_Cortex-A55x4+Cortex-A75x1 ..... 537
13.24 FVP_Base_Cortex-A55x4+Cortex-A75x2 ..... 557
13.25 FVP_Base_Cortex-A55x4+Cortex-A75x4 ..... 577
13.26 FVP_Base_Cortex-A55x4+Cortex-A76x2 ..... 598
13.27 FVP_Base_Cortex-A57x1 ..... 619
13.28 FVP_Base_Cortex-A57x1-A35x1 ..... 638
13.29 FVP_Base_Cortex-A57x1-A53x1 ..... 657
13.30 FVP_Base_Cortex-A57x2 ..... 676
13.31 FVP_Base_Cortex-A57x2-A35x4 ..... 695
13.32 FVP_Base_Cortex-A57x2-A53x4 ..... 716
13.33 FVP_Base_Cortex-A57x4 ..... 736
13.34 FVP_Base_Cortex-A57x4-A35x4 ..... 756
13.35 FVP_Base_Cortex-A57x4-A53x4 ..... 777
13.36 FVP_Base_Cortex-A65 ..... 798
13.37 FVP_Base_Cortex-A65AE ..... 818
13.38 FVP_Base_Cortex-A65AE+Cortex-A76AE ..... 838
13.39 FVP_Base_Cortex-A65AEx2+Cortex-A76AEx2 ..... 857
13.40 FVP_Base_Cortex-A65AEx4+Cortex-A76AEx4 ..... 877
13.41 FVP_Base_Cortex-A710 ..... 897
13.42 FVP_Base_Cortex-A715 ..... 920
13.43 FVP_Base_Cortex-A720 ..... 943
13.44 FVP_Base_Cortex-A72x1 ..... 967
13.45 FVP_Base_Cortex-A72x1-A53x1 ..... 985

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

13.46 FVP_Base_Cortex-A72x2 ..... 1004
13.47 FVP_Base_Cortex-A72x2-A53x4 ..... 1023
13.48 FVP_Base_Cortex-A72x4 ..... 1044
13.49 FVP_Base_Cortex-A72x4-A53x4 ..... 1063
13.50 FVP_Base_Cortex-A73x1 ..... 1085
13.51 FVP_Base_Cortex-A73x1-A53x1 ..... 1103
13.52 FVP_Base_Cortex-A73x2 ..... 1122
13.53 FVP_Base_Cortex-A73x2-A53x4 ..... 1141
13.54 FVP_Base_Cortex-A73x4 ..... 1162
13.55 FVP_Base_Cortex-A73x4-A53x4 ..... 1181
13.56 FVP_Base_Cortex-A75 ..... 1203
13.57 FVP_Base_Cortex-A76 ..... 1222
13.58 FVP_Base_Cortex-A76AE ..... 1242
13.59 FVP_Base_Cortex-A77 ..... 1262
13.60 FVP_Base_Cortex-A78 ..... 1282
13.61 FVP_Base_Cortex-A78AE ..... 1302
13.62 FVP_Base_Cortex-A78C ..... 1322
13.63 FVP_Base_Cortex-X1 ..... 1342
13.64 FVP_Base_Cortex-X1C ..... 1362
13.65 FVP_Base_Cortex-X2 ..... 1382
13.66 FVP_Base_Cortex-X3 ..... 1405
13.67 FVP_Base_Cortex-X4 ..... 1428
13.68 FVP_Base_Neoverse-E1 ..... 1452
13.69 FVP_Base_Neoverse-N1 ..... 1472
13.70 FVP_Base_Neoverse-N2 ..... 1492
13.71 FVP_Base_Neoverse-N2x1-Neoverse-N2x1 ..... 1511
13.72 FVP_Base_Neoverse-V1 ..... 1530
13.73 FVP_Base_Neoverse-V2x1-Neoverse-V2x1 ..... 1550
13.74 FVP_Base_RevC-2xAEMvA ..... 1569

**14. BaseR Platform FVPs ..... 1593**
14.1 FVP_BaseR_Cortex-R52Plus ..... 1593
14.2 FVP_BaseR_Cortex-R52x1 ..... 1612
14.3 FVP_BaseR_Cortex-R52x2 ..... 1631
14.4 FVP_BaseR_Cortex-R52x4 ..... 1650
14.5 FVP_BaseR_Cortex-R82R1 ..... 1669

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

14.6 FVP_BaseR_Cortex-R82x1 ..... 1690
14.7 FVP_BaseR_Cortex-R82x2 ..... 1709
14.8 FVP_BaseR_Cortex-R82x4 ..... 1728

**15. VE Platform FVPs ..... 1748**
15.1 FVP_VE_Cortex-A15x1 ..... 1748
15.2 FVP_VE_Cortex-A15x1-A7x1 ..... 1764
15.3 FVP_VE_Cortex-A15x2 ..... 1781
15.4 FVP_VE_Cortex-A15x2-A7x2 ..... 1797
15.5 FVP_VE_Cortex-A15x4 ..... 1815
15.6 FVP_VE_Cortex-A15x4-A7x4 ..... 1832
15.7 FVP_VE_Cortex-A17x1 ..... 1851
15.8 FVP_VE_Cortex-A17x1-A7x1 ..... 1867
15.9 FVP_VE_Cortex-A17x2 ..... 1884
15.10 FVP_VE_Cortex-A17x4 ..... 1901
15.11 FVP_VE_Cortex-A17x4-A7x4 ..... 1918
15.12 FVP_VE_Cortex-A5x1 ..... 1937
15.13 FVP_VE_Cortex-A5x2 ..... 1953
15.14 FVP_VE_Cortex-A5x4 ..... 1969
15.15 FVP_VE_Cortex-A7x1 ..... 1986
15.16 FVP_VE_Cortex-A7x2 ..... 2002
15.17 FVP_VE_Cortex-A7x4 ..... 2018
15.18 FVP_VE_Cortex-A9x1 ..... 2035
15.19 FVP_VE_Cortex-A9x2 ..... 2051
15.20 FVP_VE_Cortex-A9x4 ..... 2067
15.21 FVP_VE_Cortex-R4 ..... 2084
15.22 FVP_VE_Cortex-R5x1 ..... 2099
15.23 FVP_VE_Cortex-R5x2 ..... 2115
15.24 FVP_VE_Cortex-R7x1 ..... 2130
15.25 FVP_VE_Cortex-R7x2 ..... 2145
15.26 FVP_VE_Cortex-R8x1 ..... 2160
15.27 FVP_VE_Cortex-R8x2 ..... 2175
15.28 FVP_VE_Cortex-R8x4 ..... 2190

**16. MPS2 Platform FVPs ..... 2206**
16.1 FVP_MPS2_AEMv8M ..... 2206
16.2 FVP_MPS2_Cortex-M0 ..... 2225

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

16.3 FVP_MPS2_Cortex-M0plus ..... 2244
16.4 FVP_MPS2_Cortex-M23 ..... 2264
16.5 FVP_MPS2_Cortex-M3 ..... 2283
16.6 FVP_MPS2_Cortex-M33 ..... 2302
16.7 FVP_MPS2_Cortex-M35P ..... 2321
16.8 FVP_MPS2_Cortex-M4 ..... 2341
16.9 FVP_MPS2_Cortex-M55 ..... 2360
16.10 FVP_MPS2_Cortex-M7 ..... 2380
16.11 FVP_MPS2_Cortex-M85 ..... 2399
16.12 FVP_MPS2_SSE-200_Cortex-M33 ..... 2419
16.13 FVP_MPS2_SSE-200_Cortex-M55 ..... 2438

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

# 1. Introduction

This document describes how to get started with Fixed Virtual Platforms (FVPs). It also provides a reference for FVPs for Reference Designs, Arm Architecture FVPs, and FVPs that are included in the FVP library.

## 1.1 Conventions

The following subsections describe conventions used in Arm documents.

### Glossary

The Arm® Glossary is a list of terms used in Arm documentation, together with definitions for those terms. The Arm Glossary does not contain terms that are industry standard unless the Arm meaning differs from the generally accepted meaning.

See the Arm Glossary for more information: developer.arm.com/glossary.

| Convention | Use |
|---|---|
| *italic* | Citations. |
| **bold** | Terms in descriptive lists, where appropriate. |
| `monospace` | Text that you can enter at the keyboard, such as commands, file and program names, and source code. |
| `monospace underline` | A permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name. |
| `<and>` | Encloses replaceable terms for assembler syntax where they appear in code or code fragments.<br><br>For example:<br><br>`MRC p15, 0, <Rd>, <CRn>, <CRm>, <Opcode_2>` |
| SMALL CAPITALS | Terms that have specific technical meanings as defined in the *Arm® Glossary*. For example, **IMPLEMENTATION DEFINED**, **IMPLEMENTATION SPECIFIC**, **UNKNOWN**, and **UNPREDICTABLE**. |

Caution

Recommendations. Not following these recommendations might lead to system failure or damage.

Warning

Requirements for the system. Not following these requirements might result in system failure or damage.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

| | |
|---|---|
| Danger | Requirements for the system. Not following these requirements will result in system failure or damage. |
| Note | An important piece of information that needs your attention. |
| Tip | A useful tip that might make it easier, better or faster to perform a task. |
| Remember | A reminder of something important that relates to the information you are reading. |

## 1.2 Other information

See the Arm® website for other relevant information.

- Arm® Developer.
- Arm® Documentation.
- Technical Support.
- Arm® Glossary.

## 1.3 Useful resources

This document contains information that is specific to this product. See the following resources for other useful information.

Access to Arm documents depends on their confidentiality:

- Non-Confidential documents are available at developer.arm.com/documentation. Each document link in the following tables goes to the online version of the document.
- Confidential documents are available to licensees only through the product package.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

| Arm® product resources | Document ID | Confidentiality |
|---|---|---|
| *Arm® Architecture Models* | - | Non-Confidential |
| *Arm® Corstone™ SSE-300 Example Subsystem Technical Reference Manual* | 101773 | Non-Confidential |
| *Arm® Corstone 1000* | - | Non-Confidential |
| *Arm® Corstone™-1000 Technical Overview* | 102360 | Non-Confidential |
| *Arm® Development Platforms wiki* | - | Non-Confidential |
| *Arm® Development Studio Getting Started Guide* | 101469 | Non-Confidential |
| *Arm® Development Studio User Guide* | 101470 | Non-Confidential |
| *Arm® MPS3 FPGA Prototyping Board Technical Reference Manual* | 100765 | Non-Confidential |
| *Arm® Neoverse™ v1 reference design Software Developer Guide* | PJDOC-1779577084-33214 | Non-Confidential |
| *Corstone™-300* | - | Non-Confidential |
| *Fast Models Reference Guide* | 100964 | Non-Confidential |
| *Fast Models User Guide* | 100965 | Non-Confidential |
| *Getting started with your FVP* | - | Non-Confidential |
| *Iris Python Debug Scripting User Guide* | 101421 | Non-Confidential |
| *Iris User Guide* | 101196 | Non-Confidential |
| *Juno r2 Arm® Development Platform SoC Technical Reference Manual* | DDI 0515 | Non-Confidential |
| *Model Debugger for Fast Models User Guide* | 100968 | Non-Confidential |
| *Neoverse™ Reference Design* | - | Non-Confidential |

| Arm® architecture and specifications | Document ID | Confidentiality |
|---|---|---|
| *Arm® Architecture Reference Manual for A-profile architecture* | DDI 0487 | Non-Confidential |
| *Semihosting for AArch32 and AArch64 specification* | - | Non-Confidential |

Note

Arm tests its PDFs only in Adobe Acrobat and Acrobat Reader. Arm cannot guarantee the quality of its documents when used with any other PDF reader.

Adobe PDF reader products can be downloaded at http://www.adobe.com.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

# 2. Introduction to FVPs

*Fixed Virtual Platforms* (FVPs) enable software development without the need for real hardware.

They are available for Linux and Windows hosts, either:

- As source code examples in the Fast Models package, with the tools required to customize and build them. For more information, see the *Fast Models Reference Guide*.
- A subset of FVPs are also supplied as pre-built executables. Their composition is fixed, but you can configure their behavior using parameters.

FVPs are available for all Cortex®-A, Cortex®-R, Cortex®-M, and Neoverse™ processors, and most of them support the CADI, MTI, and Iris interfaces, so can be used for debugging and for trace output.

The scope of this document is the following pre-built FVPs, which are downloadable from Arm Developer:

- FVPs that are included in the FVP library, downloadable from Fixed Virtual Platforms.
- Arm Architecture FVPs, downloadable from Fixed Virtual Platforms.
- Arm Ecosystem FVPs, downloadable from Arm Ecosystem FVPs.

## 2.1 Types of FVP

Most Fast Models FVPs are provided in a single license-managed library. In addition, some Architecture Envelope Model (AEM) FVPs are available for download without requiring a license.

Fast Models FVPs are based on the following platforms:

- Base Platform.
- BaseR Platform.
- Arm® Versatile™ Express development boards.
- Arm® MPS2 or Arm® MPS2+ platforms, for Cortex®-M series processors.

FVPs can be obtained in the following ways:

- License-managed FVPs, including some plug-ins and utilities, are provided in the FVP Standard Library. It is available for Windows or Linux hosts. For information on how to download it, see Fixed Virtual Platforms on Arm Developer.
- Free-of-charge AEM FVPs can be downloaded from Arm Architecture Models on Arm Developer without a license. They are available for Linux hosts only. The following FVPs are available:
    - Foundation Platform. This is a basic FVP with a minimal peripheral set. It includes a single cluster which can be configured with 1-4 AEMvA cores. It is suitable for running bare-metal

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

applications and for booting Linux. It supports the CADI debug interface, but does not support MTI or Iris interfaces. It is documented in the Foundation Platform User Guide.

- Armv-A Base Platform RevC FVP. This is a platform model with a more extended peripheral set than the Foundation Platform. It has two clusters, each of which can be configured with 1-4 AEMvA cores. It supports Arm®v8-A architecture versions up to v8.7 and Armv9-A. It also supports CADI, MTI, and Iris debug and trace interfaces.
- Arm®v8-R AEM FVP. This FVP includes a single cluster of 1-4 AEMv8-R cores. It allows you to target AArch32 or AArch64, RAS, VFP, EL2, and other Arm®v8-R features.
- Armv-A Compliance FVP. This FVP is optimized for validating CPU implementations and can be used with the A-profile Architecture Compliance Kit (ACK) to demonstrate compliance with the Arm® architecture specification.

- Free-of-charge Arm Ecosystem FVPs, including Reference Design FVPs and the Morello Platform FVP can be downloaded from Arm Ecosystem FVPs on Arm Developer.

---

**Note**

Arm provides validated Linux and Android deliverables for FVPs. For details, see the Arm Development Platforms wiki on Arm Community. To get started with Linux on FVPs, see FVPs on Arm Community.

---

## Related information

Contents of the FVP Standard Library package on page 19
Base Platform
Microcontroller Prototyping System 2
Versatile Express Model

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

# 3. Getting Started with Fixed Virtual Platforms

This chapter describes how to use FVPs.

## 3.1 Requirements for FVPs

FVPs can run on a Windows or Linux host machine.

### Host machine

**Architecture**

x86-64 and Arm® AArch64 host platforms are supported.

**Minimum specification**

At least 2GB RAM, preferably 4GB.

2GHz Intel Core2Duo, or similar, that supports the MMX, SSE, SSE2, SSE3, and SSSE3 instruction sets.

**Recommended specification**

At least double the RAM of the platform you intend to simulate. For example, a simulated platform containing 8GB of DRAM should be run on a 16GB host machine.

Fast Models and associated FVPs benefit most from high single-threaded performance. For example, a high frequency (4-5GHz) Intel Core i9 or i7 or AMD Ryzen 9 or 7 host CPU gives a significant improvement, between 30-60%, over Intel Xeon cores (2-3 GHz).

### Operating system

**Linux**

Red Hat Enterprise Linux 7 or 8 (for 64-bit architectures), Ubuntu 18.04, 20.04, or 22.04 Long Term Support (LTS).

**Windows**

Microsoft Windows 10 64-bit.

### Compiler

FVPs are built with Visual Studio 2019 and GCC 9.3.0.

### Licensing

Fast Models and FVPs use either User-Based Licensing or FlexNet Licensing. Arm user-based licensing is only available to customers with a user-based licensing license. Documentation for user-based licensing is available at https://lm.arm.com. For assistance with user-based licensing issues, visit https://developer.arm.com/support and open a support case.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

For FlexNet Publisher node-locked or floating licensing, the latest version of the FlexNet software is available for download from License Server Management Software.

## 3.2 Contents of the FVP Standard Library package

The FVP Standard Library consolidates commonly used FVPs into a single package which also contains some useful plug-ins and utilities.

These FVPs are documented in the chapters 13. Base Platform FVPs on page 102, 14. BaseR Platform FVPs on page 1593, 15. VE Platform FVPs on page 1748, and 16. MPS2 Platform FVPs on page 2206.

Note

- The FVPs are supplied as pre-built executables. For the source code for these and other FVPs, install the main Fast Models product.
- The package does not include unlicensed FVPs. These are available for download separately, from Arm Architecture Models on Arm Developer.

The package installs the FVPs under the `models` directory. It also installs:

**Plug-ins**

Plug-ins are DLLs or shared objects that provide extra functionality to FVPs, for instance different types of trace output. To load a plug-in, pass the name of the plug-in to the FVP at startup using the `--plugin` command-line option or using the `FM_TRACE_PLUGINS` environment variable. For more information about plug-ins, see Plug-ins for Fast Models in the *Fast Models Reference Guide*.

**Model Shell and Model Debugger**

Model Shell is a command-line tool for launching FVPs. For more information about Model Shell, see Model Shell for Fast Models Reference Guide. Model Debugger is an easy to use symbolic debugger with a GUI that allows you to debug software running on the FVP. For more information about Model Debugger, see Model Debugger for Fast Models User Guide. They are installed in the `bin` directory.

**iris.debug Python module**

iris.debug is a Python scripting interface to Fast Models. It allows you to interact with FVPs, including connecting to and configuring them, performing execution control, and accessing registers and memory. It is installed under the `Iris` directory. For more information about iris.debug, see Iris Python Debug Scripting User Guide.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

## 3.3 FVP command-line options

Specify these options when you launch an FVP from the command line. You can specify these options in any order.

**Table 3-1: CADI or Iris-related options**

| Short form | Long form | Description |
|---|---|---|
| | `--iris-connect` *`conspec`* | Start the Iris server according to the connection specification *`conspec`*. *`conspec`* is a structured string argument which can contain flags and parameters.<br><br>Use `--iris-connect help` to print a list and description of all available Iris connection types.<br><br>The following options are ignored when using `--iris-connect`:<br>• `--iris-server`<br>• `--iris-allow-remote`<br>• `--iris-port`<br>• `--iris-port-range`<br><br>To set these connection parameters, use:<br><br>`--iris-connect tcpserver[,port=PORT][,endport=ENDPORT][,allowRemote]`<br><br>This command starts the TCP server on the first free port in the range *`PORT`*-*`ENDPORT`*, where the default for *`PORT`* is 7100 and the default for *`ENDPORT`* is *`PORT`* + 63. Only local connections are allowed, unless `allowRemote` is specified.<br><br>The other supported connection type is `socketfd=`*`FD`* which uses the socket file descriptor *`FD`* as an established UNIX domain socket connection.<br><br>Use `--iris-connect verbose=`*`n`* to set the logging level of the IrisTcpServer, where*`n`* is 0-3. |
| `-S` | `--cadi-server` | Start a CADI server. This option allows a CADI-enabled debugger to connect to targets in the simulation. To shut down the server, return to the command window that you used to start the model and press **Ctrl+C**. |
| `-I` | `--iris-server` | Start an Iris server. This option allows an Iris-enabled debugger to connect to targets in the simulation.<br><br>**Note:**<br>This option is deprecated. Use `--iris-connect` instead. |
| `-R` | `--run` | Run the simulation immediately after the CADI or Iris server is started.<br><br>Use this option with `--cadi-server` or `--iris-server`.<br><br>The default is to wait until the debugger has connected before running. |
| `-L` | `--cadi-log` | Log all CADI function calls made during the simulation into XML files.<br><br>One log file is created for each CADI target. The log files are created in the current working directory.<br><br>The filename format is:<br><br>`CADIlog-<`*`TargetInstanceName`*`>-<`*`ProcessID`*`>.xml` |

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

| Short form | Long form | Description |
|---|---|---|
| `-i` | `--iris-log` | Log to stdout all Iris function calls that were made during the simulation.<br><br>There are 5 possible log levels. To set a level greater than 1, specify the option multiple times, for example `-ii` for level 2.<br><br>The log levels have the following meanings:<br>**0**<br>Logging is disabled. This value is the default.<br>**1**<br>Log messages use a compact single-line format.<br>**2**<br>Log messages use a single-line pseudo-JSON format.<br>**3**<br>Log messages use a more readable, multi-line pseudo-JSON format.<br>**4**<br>As 3 but also prints the U64JSON hex value of the message.<br><br>**Note:**<br>To set the Iris log level for all FVP invocations, use the `IRIS_GLOBAL_INSTANCE_LOG_MESSAGES` environment variable. |
| `-A` | `--iris-allow-remote` | Start an Iris server and allow connections to it from a remote workstation.<br><br>The default is disallowed.<br><br>**Note:**<br>This option is deprecated. Use `--iris-connect` instead. |
| `-p` | `--print-port-number` | Print the port number on which the Iris or CADI server is listening.<br><br>Use this option with `--iris-server` or `--cadi-server`.<br><br>**Tip:**<br>This option can be useful if you need to specify the port number when you connect a client to the debug server. |
| | `--iris-port` *n* | Set a port to use for the Iris server.<br><br>Use this option with `--iris-server`.<br><br>The default is 7100.<br><br>**Note:**<br>This option is deprecated. Use `--iris-connect` instead. |

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

| Short form | Long form | Description |
|---|---|---|
| | `--iris-port-range` *`min:max`* | Set the range of ports to scan when starting an Iris server. The server uses the first available port in the range.<br><br>Use this option with `--iris-server`.<br><br>**Note:**<br>This option is deprecated. Use `--iris-connect` instead. |

**Table 3-2: Output-related options**

| Short form | Long form | Description |
|---|---|---|
| | `--list-instances` | Print a list of model instances to standard output, then exit the simulation.<br><br>Use this option to help identify the correct syntax for configuration files, and to find out what instances the target supplies. |
| `-l` | `--list-params` | Print a list of the available model parameters and their default values to standard output, then exit the simulation.<br><br>**Tip:**<br>If you are loading a plug-in, this option also lists the plug-in parameters. |
| | `--list-set-params` *`filename`* | Save a list of the set model parameter values for each component to a file. If *`filename`* is `-`, print to standard output instead.<br><br>An alternative to this command-line option is to use the `FASTSIM_LOGPARAMS` and `FASTSIM_LOGPARAMS_FILE` environment variables:<br>• To log parameter values to stdout, set `FASTSIM_LOGPARAMS` to 1.<br>• To log parameter values to a file, additionally set `FASTSIM_LOGPARAMS_FILE` to the name of the file. |
| | `--dump-params` | Dump the list of model parameters into a JSON file called `parameter_list.json`, then exit the simulation. The file is created in the current working directory. |
| | `--list-regs` | Print model register information to standard output, then exit the simulation. |
| | `--check-regs` | Same as `--list-regs` but with extra consistency checks on the CADI register API. |
| `-o` | `--output` *`filename`* | Redirect output from the `--list-instances`, `--list-memory`, `--list-params`, and `--list-regs` commands to a file.<br><br>If this option is used with `--list-params`, the contents of the output file are formatted correctly for use as input by the `--config-file` option. |
| | `--log` *`filename`* | Log all SystemC reports into *`filename`*. |

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

| Short form | Long form | Description |
|---|---|---|
| | `--stat` | Print the following performance statistics on simulation exit:<br>**Simulated time**<br>An estimate of the time that the workload would have taken on the modeled hardware.<br>**User time**<br>Time in wall clock seconds that the host CPU spent running in user mode.<br>**System time**<br>Time in wall clock seconds that the host CPU spent running in system mode.<br>**Wall time**<br>Time in wall clock seconds between the simulation starting and stopping.<br>**Performance index**<br>An estimate of the accuracy of the simulation performance. This value is Simulated time divided by Wall time. |
| `-P` | `--prefix` | Prefix each line of semihosting output with the name of the target instance. |
| `-h` | `--help` | Print the help message and exit. |
| | `--version` | Print version information for the FVP. |
| `-q` | `--quiet` | Suppress informational output. |
| `-K` | `--keep-console` | Keep the console window open after completion. This option applies to Windows hosts only. |
| | `--disable-model-exitcode` | Disable the simulation from retrieving the exit code returned by a model or a plug-in. By default, it is enabled. |

**Table 3-3: Run control options**

| Short form | Long form | Description |
|---|---|---|
| | `--cpulimit` *n* | Maximum number of wall-clock seconds for the simulation process to be active. This value excludes simulation startup and shutdown.<br><br>This option is ignored if a debug server is started.<br><br>The default is unlimited. |
| | `--cyclelimit` *n* | Maximum number of cycles to run.<br><br>This option is ignored if a debug server is started.<br><br>The default is unlimited. |
| `-T` | `--timelimit` *n* | Maximum number of wall-clock seconds for the simulation to run, excluding startup and shutdown. To terminate the model immediately after initialization, specify `--timelimit 0`. |
| | `--simlimit` *n* | Maximum number of seconds to simulate.<br><br>This option is ignored if a debug server is started.<br>The default is unlimited.<br><br>Like the `Simulated time` value output by `--stat`, this value is measured in simulation seconds, not wall-clock seconds. |

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

| Short form | Long form | Description |
|---|---|---|
| `-b` | `--break [instance=] [threadid:][memspace@] address` | Set a program breakpoint on the address of an instruction, where:<br>• `threadid` is an optional thread id, for a thread-specific breakpoint.<br>• `memspace` is an optional name or id of the memory space that `address` is in. If not specified, the breakpoint is set on the first program memory space found.<br>• If the FVP has multiple cores, you must specify an instance, for example:<br>`-b FVP_Base_AEMvA.cluster0.cpu0=0x800010eC`<br>This option can be specified multiple times. |

**Table 3-4: Timing and performance options**

| Short form | Long form | Description |
|---|---|---|
| | `--cpi-file filename` | Use *`filename`* to set the Cycles Per Instruction (CPI) class.<br>For information about CPI files, see Timing Annotation. |
| `-Q` | `--quantum n` | Number of ticks to simulate for each quantum. The default is 10000. |
| `-M` | `--min-sync-latency n` | Number of ticks to simulate before synchronizing. Events that occur at a higher frequency than this value are missed. The default is 100. |
| | `--fast-ram filename` | Enable FastRAM and load the configuration from *`filename`*. For more information about FastRAM, see FastRAM. |

**Table 3-5: Memory-related options**

| Short form | Long form | Description |
|---|---|---|
| | `--dump file@address,size` | Dump a section of memory to a file at model shutdown. This option can be specified multiple times. The full syntax is:<br>`--dump [instance=]file@[memspace:]address,size`<br>**Tip:**<br>To see the list of instances and memory spaces, use the `--list-memory` option. |
| | `--data file@address` | Write raw data contained in *`file`* to the specified address. This option can be specified multiple times. The full syntax is:<br>`--data [instance=]file@[memspace:]address` |
| | `--list-memory` | Print model memory information to standard output, then exit the simulation. |
| | `--start [instance=] address` | Set the initial PC value to this address, overriding the `.axf` start address.<br>**Note:**<br>• Use this option if you do not want the CPU to start executing at the default reset address. You do not normally need to do this if you are loading an ELF file using `--application`.<br>• This option can be used with `--data` to load binary data that is not in an ELF file. |

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**Table 3-6: Configuration options**

| Short form | Long form | Description |
|---|---|---|
| `-C` | `--parameter` *`instance.parameter=value`* | Set a parameter. This option can be specified multiple times. Specify the full hierarchical name of the parameter.<br><br>This option is also used to set plug-in parameters. |
| `-f` | `--config-file` *`filename`* | Load the parameters from a configuration file. |

**Table 3-7: Options for loading a plug-in or application**

| Short form | Long form | Description |
|---|---|---|
| `-a` | `--application` [*`instance`*=] *`filename`* | Load an application.<br><br>On a multi-core system, specify the instance, or use `*` to load the application image into all the cores in a cluster:<br><br>`-a cluster0.cpu*=`*`file`* |
| | `--plugin` *`filename`* | Load the plug-in *`filename`*. This option can be specified multiple times. You can also load plug-ins using the `FM_TRACE_PLUGINS` environment variable. For information about plug-ins, see Plug-ins for Fast Models. |
| | `--trace-plugin` *`filename`* | Load a trace plug-in.<br><br>**Note:**<br>This option is deprecated. Use `--plugin` instead. |

## 3.4 Loading and running an application on an FVP

There are different ways to launch an FVP, for example from the command prompt, or from Model Debugger or Arm® Development Studio.

To run an FVP from the command prompt, enter the model name followed by the model options. To see all available options, use the `--help` option. This is a list of some of the commonly used options for FVPs:

**-a [*instance*=]*filename*.axf**

Specifies an application to load, and optionally, the instance or instances to load it on. The file can be in one of the following formats, or in a gzip-compressed version:

- ELF.
- Motorola S-Record.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

If the FVP contains multiple core instances, you can specify the instance to load the image on. The instance name can include a wildcard (*) to load the same application image into multiple cores, for example:

```
./FVP_Base_AEMvA -a cluster0.cpu*=__image.axf
```

Omitting the instance name loads the application on all cores in the first cluster. If the FVP has multiple cores but no clusters, you must specify the instance name.

**--data *filename*.bin@*address***

Loads binary data into memory at the address specified.

**-C *instance.parameter=value***

Sets a single model parameter. Parameters are specified using a path that separates the instance names and the parameter using dots. For example, `-c bp.flashloader0.fname=fip.bin`. Here, `bp` and `flashloader0` are instance names and `fname` is the parameter. To set multiple parameters using a configuration file, use the `-f` option instead. To list all the available parameters, with their type, default value, and description, run the model with the `--list-params`, or `-l` option.

**-f *config_file*.txt**

Specifies the name of a plain text configuration file. Configuration files simplify managing multiple model parameters. You can set the same parameters using this option as with the `-c` option.

**-S**

Starts a CADI debug server. This option allows a CADI-enabled debugger, such as Model Debugger or Arm® Development Studio Debugger, to connect to the running model. By default, the model waits for the debugger to connect before starting.

### Related information

Arm Development Studio Getting Started Guide
Using Model Debugger

## 3.5 Configuring the model

When you start the model from the command line, you can configure it using either:

- One or more `-c` command-line arguments.
- A configuration file and the `-f` command-line argument.

Each `-c` command-line argument or line in the configuration file must contain:

- The name of the component instance.
- The parameter to modify.
- Its value.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Use the following format:

```
instance.parameter=value
```

The `instance` can be a hierarchical path, with each level separated by a dot `.` character.

---

Note

- Comment lines in the configuration file begin with a `#` character.
- You can set Boolean values using either `true` or `false`, or `1` or `0`.

---

You can generate a configuration file with all parameters set to default values by using the `-o` option to redirect the output from the `--list-params` option, for example:

```
FVP_Base_AEMvA.exe --list-params -o params.txt
```

## 3.6 FVP debug

This section describes how to debug an FVP.

### FVP debug options

To debug an FVP, you can either:

- Run the FVP from within a CADI-enabled debugger.
- Start the FVP with the `-s` command-line argument and then connect a CADI-enabled debugger to it.

For information about using your debugger in these ways, see your debugger documentation.

### Semihosting support

Semihosting enables code running on a platform model to directly access the I/O facilities on a host computer. Examples of these facilities include console I/O and file I/O.

The simulator handles semihosting by intercepting `HLT 0xF000`, `SVC 0x123456`, or `SVC 0xAB`, depending on whether the processor is in A64, A32 or T32 state. It handles all other `HLT`s and `SVC`s as normal.

If the operating system does not use `HLT 0xF000`, `SVC 0x123456`, or `SVC 0xAB` for its own purposes, it is not necessary to disable semihosting support to boot an operating system.

To temporarily or permanently disable semihosting support for a current debug connection, see your debugger documentation.

### Related information

Semihosting for AArch32 and AArch64

Using semihosting to access resources on the host computer

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

# 3.7 Using the CLCD window

When an FVP starts, the CLCD window opens, representing the contents of the simulated color LCD frame buffer. It automatically resizes to match the horizontal and vertical resolution that is set in the CLCD peripheral registers.

## MPS2 FVPs

The LEDs and MCC switches in the CLCD window for MPS2-based FVPs correspond to the LEDs and switches on the physical board. They are controlled by the software that you load onto the FVP. For information on how to use them, see User switches and user LEDs in the Arm® MPS2 and MPS2+ FPGA Prototyping Boards TRM on Arm Developer.

## Base Platform and VE FVPs

The top section of the CLCD window for Base Platform and VE FVPs displays the status information:

**Figure 3-1: Base Platform CLCD window in its default state at startup**

**USERSW**

Eight white boxes show the state of the User DIP switches.

These represent switch S6 on the VE hardware, USERSW[8:1], which is mapped to bits [7:0] of the SYS_SW register at address `0x1C010004`.

The switches are in the off position by default. To change its state, click in the area above or below a white box.

**BOOTSW**

Eight white boxes show the state of the VE Boot DIP switches.

These represent switch S8 on the VE hardware, BOOTSEL[8:1], which is mapped to bits [15:8] of the SYS_SW register at address `0x1C010004`.

The switches are in the off position by default.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Note

Changing Boot DIP switch positions while the model is running can result in unpredictable behavior.

**S6LED**

Eight colored boxes indicate the state of the VE User LEDs.

These represent the red/yellow/green LEDs on the VE hardware, which are mapped to bits [7:0] of the SYS_LED register at address `0x1C010008`.

**Daughter**

Eight white boxes show the state of the daughterboard DIP switches and eight colored boxes show the state of the daughterboard LEDs.

**Total Instr**

A counter showing the total number of instructions executed.

Because the FVP models provide a *Programmer's View* (PV) of the system, the CLCD displays total instructions rather than total processor cycles. Timing might differ substantially from the hardware because:

- Bus fabric is simplified.
- Memory latencies are minimized.
- Cycle approximate processor and peripheral models are used.

In general, bus transaction timing is consistent with the hardware, but the timing of operations within the model is not accurate.

**Total Time**

A counter showing the total elapsed time, in seconds.

This time is wall clock time, not simulated time.

**Rate Limit**

A feature that disables or enables fast simulation.

Because the system model is highly optimized, your code might run faster than it would on real hardware. This effect might cause timing issues.

Rate Limit is disabled by default to favor simulation speed. Enable it to restrict simulation time so that it more closely matches real time.

To enable or disable Rate Limit, click the square button. You can also configure this option when instantiating the model with the `rate_limit-enable` visualization component parameter.

When you click the **Total Instr** item in the CLCD, the display toggles to show the following:

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**Instr/sec**

The number of instructions that execute per second of wall clock time.

**Perf Index**

The ratio of real time to simulation time. The larger the ratio, the faster the simulation runs. If you enable the Rate Limit feature, the Perf Index approaches unity.

You can reset the simulation counters by resetting the model.

The FVP CLCD displays the core run state for each core on each cluster using a colored icon. The icons are to the left of the **Total Instr** (or **Inst/sec**) item.

**Figure 3-2: Core run state icons for a dual-cluster, quad-core model**

**Table 3-8: Core run state icon descriptions**

| Icon | State label | Description |
|---|---|---|
| | UNKNOWN | Run status unknown, that is, simulation has not started. |
| | RUNNING | Core running, is not idle, and is executing instructions. |
| | HALTED | External halt signal asserted. |
| | STANDBY_WFE | Last instruction executed was WFE and standby mode has been entered. |
| | STANDBY_WFI | Last instruction executed was WFI and standby mode has been entered. |
| | IN_RESET | External reset signal asserted. |
| | DORMANT | Partial core power down. |
| | SHUTDOWN | Complete core power down. |

If the CLCD window has focus:

- Any keyboard input is translated to PS/2 keyboard data.
- Any mouse activity over the window is translated into PS/2 relative mouse motion data. The data is then streamed to the KMI peripheral model FIFOs.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Note

The simulator only sends relative mouse motion events to the model. As a result, the host mouse pointer does not necessarily align with the target OS mouse pointer.

You can hide the host mouse pointer by pressing the **left Ctrl**+**left Alt** keys. Press the keys again to redisplay the host mouse pointer. Only the **left Ctrl** key is operational. The **right Ctrl** key does not have the same effect.

If you prefer to use a different key, configure it with the `trap_key` visualization component parameter.

### Related information

VEVisualisation component

## 3.8 Ethernet with VE FVPs

This section describes how to use Ethernet with VE FVPs.

### Using Ethernet with VE FVPs

The VE FVPs have a virtual Ethernet component. This component is a model of the SMSC 91C111 Ethernet controller, and uses a TAP device to communicate with the network. By default, the Ethernet component is disabled.

### Host requirements

Before you can use the Ethernet capability of VE FVPs, set up your host computer.

### Target requirements

This section describes the target requirements.

**Target requirements - about**

The VE FVPs include a software implementation of the SMSC 91C111 Ethernet controller. Your target OS must therefore include a driver for this specific device. To use the SMSC chip, configure the kernel. Linux supports the SMSC 91C111.

The configurable SMSC 91C111 component parameters are:

- `enabled`.
- `mac_address`.
- `promiscuous`.

**enabled**

When the device is disabled, the kernel cannot detect the device.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**Figure 3-3: Model networking structure block diagram**

To perform read and write operations on the TAP device, configure a HostBridge component. The HostBridge component is a virtual *Programmer's View* (PV) model. It acts as a networking

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

gateway to exchange Ethernet packets with the TAP device on the host, and to forward packets to NIC models.

**mac_address**

There are two options for the `mac_address` parameter.

If a MAC address is not specified, when the simulator is run it takes the default MAC address, which is randomly generated. This random generation provides some degree of MAC address uniqueness when running models on multiple hosts on a local network.

**promiscuous**

The Ethernet component starts in promiscuous mode by default. In this mode, it receives all network traffic, even any not addressed to the device. Use this mode if you are using a single network device for multiple MAC addresses. Use this mode if, for example, you share the network card between your host OS and the VE FVP Ethernet component.

By default, the Ethernet device on the VE FVP has a randomly generated MAC address and starts in promiscuous mode.

## 3.9 Using a terminal with a system model

The TelnetTerminal component is a virtual component that enables UART data to be transferred between a TCP/IP socket on the host and a serial port on the target.

### Using TelnetTerminal

The following figure shows a block diagram of one possible relationship between the target and host through the TelnetTerminal component. The TelnetTerminal block is what you configure when you define Terminal component parameters. The Virtual Machine is an FVP.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**Figure 3-4: Terminal block diagram**

On the target side, the console process that is invoked by your target OS relies on a suitable driver being present. Such drivers are normally part of the OS kernel. The driver passes serial data through a UART. The data is forwarded to the TelnetTerminal component. When the simulation is started and the TelnetTerminal component is enabled, the component opens a server (listening) socket on a TCP/IP port. This is port 5000 by default. This port can be connected to by, for example, a Telnet process on the host.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

When data becomes available on the network socket, the TelnetTerminal component buffers the data, which can then be read from SerialData.

If there is no connection to the network socket when the first data access is made, and the `start_telnet` parameter is true, a host Telnet session is started automatically. Prior to this first access, you can connect a client of your choice to the network socket.

If the connection between the TelnetTerminal component and the client is broken at any time, for example by closing a client Telnet session, the port is re-opened on the host, permitting you to make another client connection. This could have a different port number if the original one is no longer available.

The port number of a particular TelnetTerminal instance can be defined when your model system starts. The actual value of the port used by each TelnetTerminal is declared when it starts or restarts, and might not be the value that you specified if the port is already in use. If you are using Model Shell, the port numbers are displayed in the host window in which you started the model.

---

Note

Microsoft Windows 10 disables the Telnet client by default. Follow these steps to enable it:

1. Select **Start > Settings**.
2. In the search box, type **Turn Windows features on or off**. The **Windows Features** dialog opens.
3. Select the **Telnet Client** check box and click **OK**. The installation might take several minutes to complete.

---

## TelnetTerminal parameters

To set the parameters, the syntax to use in a configuration file or on the command line is:

```
motherboard.terminal_x.parameter=value
```

where *x* is the terminal identifier and can be 0, 1, 2, or 3.

You can start the TelnetTerminal component in either of the following modes, depending on the `mode` parameter:

**`telnet`**

In Telnet mode, the terminal component supports a subset of the RFC 854 protocol. This means that the terminal participates in negotiations between the host and client concerning what is and is not supported, but there is no flow control.

**`raw`**

In raw mode the byte stream passes unmodified between the host and the target. The terminal does not participate in initial capability negotiations between the host and client. Instead it acts as a TCP/IP port. You can use this feature to directly connect to your target through the TelnetTerminal component. This permits a debugger connection, for example, to connect a gdb client to a gdbserver running on the target operating system.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

The `terminal_command` parameter specifies the command line used to launch a terminal application and connect to the opened TCP port. The TelnetTerminal component replaces the keywords `%port` and `%title`, if specified, with the opened port number and component name, respectively. After replacing `%port` and `%title`, the command line is executed verbatim.

An empty string, which is the default, launches xterm on Linux or `telnet.exe` on Windows.

---

Note

If you specify a non-empty string, it must include `%port`, but `%title` is optional.

---

For example:

```
fvp_mps2.telnetterminal0.terminal_command="putty.exe -telnet localhost %port"
```

## 3.10 Virtio P9 device component

The VirtioP9Device component is included in Base, BaseR, and A-profile VE platforms. It implements a subset of the Plan 9 file protocol over a virtio transport. It enables accessing a directory on the host's filesystem within Linux, or another operating system that implements the protocol, running on a platform model.

Take the following steps to set up this component:

- Use a version of Linux that supports v9fs over virtio and virtio-mmio devices.
- Update the device tree to include the VirtioP9Device component, or specify it on the kernel command-line, as shown below. The address range for both VE and Base platforms is `0x1C140000-0x1C14FFFF`.
  The interrupt number is 43, or IRQ 75, for both VE and Base platforms.
- Set the following parameter to the directory on the host that you want to mount in the model:

  **VE:**

  ```
  motherboard.virtiop9device.root_path
  ```

  **Base:**

  ```
  bp.virtiop9device.root_path
  ```
- On Linux, mount the host directory by using the following command in the model:

  ```
  $ mount -t 9p -o trans=virtio,version=9p2000.L FM <mount point>
  ```

Example kernel command-line argument:

```
virtio_mmio.device=0x10000@0x1c140000:75
```

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Example entry for DTS files, to add next to the corresponding `virtio_block` entry:

```
virtio_p9@0140000 {
       compatible = "virtio,mmio";
       reg = <0x0 0x1c140000 0x0 0x1000>;
       interrupts = <0x0 0x2b 0x4>;
};
```

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

# 4. Arm® CoreLink™ SGI-575 reference design FVP

The SGI-575 FVP provides a software model of the Arm® CoreLink™ SGI-575 System Guidance for Infrastructure reference subsystem. It drives system architecture and software standardization and provides software and binaries of proprietary firmware that reduce the amount of work that is required for SoC development.

The FVP is used with the SGI-575 software package.

See the Arm® CoreLink™ SGI-575 System Guidance for Infrastructure Software Bundle Readme for instructions on how to set up and run the FVP.

## 4.1 About the SGI-575 FVP

The SGI-575 Fixed Virtual Platform (FVP) models multiple IP components.

---

Note

It does not model every component that SGI-575 describes. For example, it does not model the Arm® CoreSight™ technology components.

---

The FVP models the following IP components:

- 2 × MP4 Cortex®-A75 clusters
- SCP
- MCP
- CMN-600
- Multiple NIC-400 interconnects
- Memory access path towards DRAM

## 4.2 SGI-575 FVP peripherals

The SGI-575 FVP includes peripherals that the software payload requires to run.

These peripherals are organized in two layers:

**SoC**

The SoC peripherals represent peripherals that are added to a compute subsystem in a SoC design.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**Board**

The board peripherals represent peripherals that are present on the board onto which the SoC is mounted.

### 4.2.1 Memory map for SGI-575 FVP SoC peripherals

This table shows the memory map for the SoC peripherals in the SGI-575 FVP.

**Table 4-1: SoC peripherals**

| Name | Base address | Size | Description |
|---|---|---|---|
| SMC interface | `0x00_0800_0000` | 368MB | Routed to Board |
| DMA MMU-400 | `0x00_7FB0_0000` | 64KB | - |
| HDLCD1 MMU-400 | `0x00_7FB1_0000` | 64KB | - |
| HDLCD0 MMU-400 | `0x00_7FB2_0000` | 64KB | - |
| SoC Interconnect NIC400 GPV | `0x00_7FD0_0000` | 1MB | - |
| Surge detector | `0x00_7FE5_0000` | 4KB | Dummy APB |
| TRNG | `0x00_7FE6_0000` | 4KB | - |
| Trusted Non-volatile counters | `0x00_7FE7_0000` | 4KB | - |
| Trusted Root-Key storage | `0x00_7FE8_0000` | 4KB | - |
| Secure I2C | `0x00_7FE9_0000` | 256B | PL061 GPIO |
| DDR4 PHY 1 | `0x00_7FB7_0000` | 64KB | Dummy APB |
| DDR4 PHY 0 | `0x00_7FB6_0000` | 64KB | Dummy APB |
| DMA Non-secure | `0x00_7FF0_0000` | 4KB | - |
| DMA Secure | `0x00_7FF1_0000` | 4KB | - |
| HDCLD1 | `0x00_7FF5_0000` | 4KB | - |
| HDCLD1 | `0x00_7FF6_0000` | 4KB | - |
| UART 1 | `0x00_7FF7_0000` | 4KB | - |
| UART 0 | `0x00_7FF8_0000` | 4KB | - |
| I2S | `0x00_7FF9_0000` | 1KB | PL061 GPIO |
| I2C | `0x00_7FFA_0000` | 256B | PL061 GPIO |
| PL352 | `0x00_7FFD_0000` | 4KB | PL354 |
| System override Registers | `0x00_7FFF_0000` | 4KB | - |
| AP configuration | `0x00_7FFE_0000` | 4KB | GPR |

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

### 4.2.2 Memory map for SGI-575 FVP board peripherals

This table shows the memory map for the board peripherals in the SGI-575 FVP.

**Table 4-2: Board peripherals**

| Name | Base address | Size | Description |
|---|---|---|---|
| NOR Flash 0 | `0x00_0800_0000` | 64MB | - |
| NOR Flash 1 | `0x00_0C00_0000` | 64MB | - |
| NOR Flash 2 | `0x00_1000_0000` | 64MB | - |
| Ethernet | `0x00_1800_0000` | 64MB | SMSC 91C111 |
| System Registers | `0x00_1C01_0000` | 64KB | - |
| MCI | `0x00_1C05_0000` | 64KB | PL180 |
| KMI 0 | `0x00_1C06_0000` | 64KB | PL050 |
| KMI 1 | `0x00_1C07_0000` | 64KB | PL050 |
| UART 0 | `0x00_1C09_0000` | 64KB | PL011 |
| UART 1 | `0x00_1C0A_0000` | 64KB | PL011 |
| VFS2 | `0x00_1C0D_0000` | 64KB | - |
| Watchdog | `0x00_1C0F_0000` | 64KB | SP805 |
| Dual Timer | `0x00_1C11_0000` | 64KB | SP084 |
| Virtio Block Device | `0x00_1C13_0000` | 64KB | - |
| RTC | `0x00_1C17_0000` | 64KB | PL031 |
| GPIO 0 | `0x00_1C1D_0000` | 64KB | - |
| GPIO 1 | `0x00_1C1E_0000` | 64KB | - |
| DRAM | `0x00_8000_0000` | 2GB | - |
| DRAM | `0x80_8000_0000` | 6GB | - |

### 4.2.3 Interrupt maps for SGI-575 FVP

These tables show the interrupt IDs and sources for the FVP peripherals.

**Table 4-3: Interrupt map at the SoC layer**

| Interrupt ID | Source | Description |
|---|---|---|
| 147 | UART 0 | - |
| 148 | UART 1 | - |
| 171 | TRNG | - |

**Table 4-4: Interrupt map at the board layer**

| Interrupt ID | Source | Description |
|---|---|---|
| 111 | Ethernet | - |
| 132 | RTC | - |
| 133 | UART 0 | - |

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

| Interrupt ID | Source | Description |
|---|---|---|
| 134 | UART 1 | - |
| 140 | VFS2 | - |
| 202 | Virtio | - |
| 228 | Watchdog | - |
| 229 | KMI 0 | - |
| 230 | Dual Timer | Interrupts 0 and 1 |
| 231 | System Registers Ethernet IRQ | - |

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

# 5. Arm® Corstone™ SSE-300 FVP

To develop ahead of hardware availability and to explore the design from a software perspective, the Corstone™ SSE-300 Fixed Virtual Platform (FVP) models much of the Arm® IP in the Corstone™ SSE-300 subsystem version r0p1.

The Corstone™ SSE-300 FVP drives system architecture and software standardization. It is used with the Corstone™ SSE-300 software package which provides software and binaries of proprietary firmware that reduce the amount of work that is required for SoC development. See the Corstone™ SSE-300 software bundle readme for instructions on how to set up and run the FVP.

The FVP models the following IP components:

- A single Arm® Cortex®-M55 processor with the MVE extension.
- A single Arm® Ethos™-U55 or Ethos™-U65 processor.
- Memory Protection Controller (MPC).
- Peripheral Protection Controller (PPC).
- Implementation Defined Attribution Unit (IDAU).

The full description of components that are internal to the SSE-300 subsystem can be found in the Arm Corstone SSE-300 Example Subsystem Technical Reference Manual.

The FVP has the following limitations:

- It does not model every component that Corstone™ SSE-300 describes. For example, it does not model the CoreSight™ technology components.
- QSPI is not modeled. It has QSPI SRAM instead.
- Some components are dummy stubs with a minimal implementation. Models with dummy APB have memory access.
- Some components are partially modeled.

Refer to the following documents for more information:

- Arm Corstone SSE-300 Example Subsystem Technical Reference Manual
- Arm MPS3 FPGA Prototyping Board Technical Reference Manual
- Corstone-300 on Arm Developer

## 5.1 Memory map overview for Corstone™ SSE-300

This table outlines the main FVP memories and their positions within the memory map.

This memory map includes IDAU security information for memory regions.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**Table 5-1: Memory map overview**

| Row ID | Address range | | Size | Description | Alias with Row ID | IDAU region values | | |
|---|---|---|---|---|---|---|---|---|
| | From | To | | | | Security | IDAUID | NSC |
| 1 | 0x0000_0000 | 0x0007_FFFF | 512KB | ITCM[3] | 5 | NS | 0 | 0 |
| 2 | 0x0008_0000 | 0x00FF_FFFF | 15.5MB | Reserved | - | | | |
| 3 | 0x0100_0000 | 0x010F_FFFF | 1MB | SRAM (only 1MB)[1] | 7 | | | |
| 4 | 0x0110_0000 | 0x0FFF_FFFF | 239MB | Reserved | - | | | |
| 5 | 0x1000_0000 | 0x1007_FFFF | 512KB | ITCM[3] | 1 | S | 1 | CODE NSC |
| 6 | 0x1008_0000 | 0x10FF_FFFF | 15.5MB | Reserved | - | | | |
| 7 | 0x1100_0000 | 0x110F_FFFF | 1MB | SRAM (only 1MB)[1] | 3 | | | |
| 8 | 0x1110_0000 | 0x1FFF_FFFF | 239MB | Reserved | - | | | |
| 9 | 0x2000_0000 | 0x2007_FFFF | 512KB | DTCM (4 x banks of 128KB)[3] | 15 | NS | 2 | 0 |
| 10 | 0x2008_0000 | 0x20FF_FFFF | 15.5MB | Reserved | - | - | - | - |
| 11 | 0x2100_0000 | 0x211F_FFFF | 2MB | Internal SRAM area (SSE-300 implements 2x1MB)[3] | 17 | - | - | - |
| 12 | 0x2120_0000 | 0x27FF_FFFF | 110MB | Reserved | - | - | - | - |
| 13 | 0x2800_0000 | 0x287F_FFFF | 8MB | QSPI (only 8MB)[1] | 19 | - | - | - |
| 14 | 0x2880_0000 | 0x2FFF_FFFF | 120MB | Reserved | - | - | - | - |
| 15 | 0x3000_0000 | 0x3007_FFFF | 512KB | DTCM (4 x banks of 128KB)[3] | 9 | S | 3 | RAM NSC |
| 16 | 0x3008_0000 | 0x30FF_FFFF | 15.5MB | Reserved | - | - | - | - |
| 17 | 0x3100_0000 | 0x311F_FFFF | 2MB | Internal SRAM area (SSE-300 implements 2x1MB)[3] | 11 | - | - | - |
| 18 | 0x3120_0000 | 0x37FF_FFFF | 110MB | Reserved | - | - | - | - |
| 19 | 0x3800_0000 | 0x387F_FFFF | 8MB | QSPI (only 8MB)[1] | 13 | - | - | - |
| 20 | 0x3880_0000 | 0x3FFF_FFFF | 120MB | Reserved | - | - | - | - |
| 21 | 0x4000_0000 | 0x47FF_FFFF | 128MB | Non-Secure Low Latency Peripheral Region. | 23 | NS | 4 | 0 |
| 22 | 0x4800_0000 | 0x4FFF_FFFF | 128MB | Non-Secure High Latency Peripheral Region. | 24 | NS | 4 | 0 |
| 23 | 0x5000_0000 | 0x57FF_FFFF | 128MB | Secure Low Latency Peripheral Region. | 21 | S | 5 | 0 |
| 24 | 0x5800_0000 | 0x5FFF_FFFF | 128MB | Secure High Latency Peripheral Region. | 22 | S | 5 | 0 |
| 25 | 0x6000_0000 | 0x6FFF_FFFF | 256MB | DDR4[1] | - | NS | 6 | 0 |
| 26 | 0x7000_0000 | 0x7FFF_FFFF | 256MB | DDR4[1] | - | S | 7 | 0 |
| 27 | 0x8000_0000 | 0x8FFF_FFFF | 256MB | DDR4[1] | - | NS | 8 | 0 |
| 28 | 0x9000_0000 | 0x9FFF_FFFF | 256MB | DDR4[1] | - | S | 9 | 0 |
| 29 | 0xA000_0000 | 0xAFFF_FFFF | 256MB | DDR4[1] | - | NS | A | 0 |
| 30 | 0xB000_0000 | 0xBFFF_FFFF | 256MB | DDR4[1] | - | S | B | 0 |
| 31 | 0xC000_0000 | 0xCFFF_FFFF | 256MB | DDR4[1] | - | NS | C | 0 |

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

| Row ID | Address range | | Size | Description | Alias with Row ID | IDAU region values | | |
|---|---|---|---|---|---|---|---|---|
| | From | To | | | | Security | IDAUID | NSC |
| 32 | `0xD000_0000` | `0xDFFF_FFFF` | 256MB | DDR4[1] | - | S | D | 0 |
| 33 | `0xE000_0000` | `0xE00F_FFFF` | 1MB | External Private Peripheral Bus | - | - | Exempt | - |
| 34 | `0xE010_0000` | `0xE01F_FFFF` | 1MB | Reserved | - | NS | E | 0 |
| 35 | `0xE020_0000` | `0xEFFF_FFFF` | 254MB | Maps to HMSTEXPPILL expansion interface[2] | - | NS | E | 0 |
| 36 | `0xF000_0000` | `0xF00F_FFFF` | 1MB | Reserved | - | - | Exempt | - |
| 37 | `0xF010_0000` | `0xF01F_FFFF` | 1MB | Reserved | - | S | F | 0 |
| 38 | `0xF020_0000` | `0xFFFF_FFFF` | 254MB | Maps to HMSTEXPPILL expansion interface[2] | - | S | F | 0 |

## 5.2 Corstone™ SSE-300 FVP peripherals

The Corstone™ SSE-300 FVP includes peripherals that the software payload requires to run.

Board peripherals are peripherals that might be present on the board onto which the SoC is mounted. The Corstone™ SSE-300 board model is based on the Arm® MPS3 board.

Note

Some of the MPS3 Fast Models have minimal implementations. For more information, refer to the documents in the FVP package.

All peripherals that are extensions to the Corstone™ SSE-300 subsystem are mapped into two key areas of the memory map:

**`0x4000_0000` to `0x47FF_FFFF` and `0x4800_0000` to `0x4FFF_FFFF`**
: Non-secure region that maps to AHB Master Expansion 1 interface.

**`0x5000_0000` to `0x57FF_FFFF` and `0x5800_0000` to `0x5FFF_FFFF`**
: Secure region that maps to AHB Master Expansion 1 interface.

[1] Security access is controlled by MPC.
[2] Accesses to these addresses result in an AHB5 error response.
[3] For security settings, control, and features, refer to the Arm® Corstone™ SSE-300 documentation.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

### 5.2.1 Corstone™ SSE-300 non-secure expansion peripherals memory map

Memory map for non-secure board peripherals.

**Table 5-2: Non-secure board peripherals**

| Row ID | Address range in non-secure region | | Size | Description | Modeled in FVP |
|---|---|---|---|---|---|
| | From | To | | | |
| 1 | 0x4110_0000 | 0x4110_0FFF | 4KB | GPIO 0 | CMSDK GPIO |
| 2 | 0x4110_1000 | 0x4110_1FFF | 4KB | GPIO 1 | CMSDK GPIO |
| 3 | 0x4110_2000 | 0x4110_2FFF | 4KB | GPIO 2 | CMSDK GPIO |
| 4 | 0x4110_3000 | 0x4110_3FFF | 4KB | GPIO 3 | CMSDK GPIO |
| 5 | 0x4110_4000 | 0x4110_4FFF | 4KB | FMC GPIO 0 | CMSDK GPIO |
| 6 | 0x4110_5000 | 0x4110_5FFF | 4KB | FMC GPIO 1 | CMSDK GPIO |
| 7 | 0x4110_6000 | 0x4110_6FFF | 4KB | FMC GPIO 2 | CMSDK GPIO |
| 8 | 0x4110_7000 | 0x4110_7FFF | 4KB | FMC USER AHB | Not modeled |
| - | 0x4110_8000 | 0x411F_FFFF | - | Reserved | - |
| 9 | 0x4120_0000 | 0x4120_0FFF | 4KB | USER AHB 0 | Not modeled |
| 10 | 0x4120_1000 | 0x4120_1FFF | 4KB | USER AHB 1 | Not modeled |
| 11 | 0x4120_2000 | 0x4120_2FFF | 4KB | USER AHB 2 | Not modeled |
| 12 | 0x4120_3000 | 0x4120_3FFF | 4KB | USER AHB 3 | Not modeled |
| - | 0x4120_4000 | 0x413F_FFFF | - | Reserved | - |
| 13 | 0x4140_0000 | 0x414F_FFFF | 1MB | Ethernet | SMSC91C111 Ethernet controller |
| 14 | 0x4150_0000 | 0x415F_FFFF | 1MB | USB | Dummy stub |
| - | 0x4160_0000 | 0x416F_FFFF | - | Reserved | - |
| 15 | 0x4170_0000 | 0x4170_0FFF | 4KB | User APB0 | Not modeled |
| 16 | 0x4170_1000 | 0x4170_1FFF | 4KB | User APB1 | Not modeled |
| 17 | 0x4170_2000 | 0x4170_2FFF | 4KB | User APB2 | Not modeled |
| 18 | 0x4170_3000 | 0x4170_3FFF | 4KB | User APB3 | Not modeled |
| - | 0x4170_4000 | 0x417F_FFFF | - | Reserved | - |
| 19 | 0x4180_0000 | 0x4180_0FFF | 4KB | QSPI Config | Not modeled |
| 20 | 0x4180_1000 | 0x4180_1FFF | 4KB | QSPI Write | Not modeled |
| - | 0x4180_2000 | 0x47FF_FFFF | - | Reserved | - |
| - | 0x4800_0000 | 0x480F_FFFF | - | Subsystem peripherals | - |
| 21 | 0x4810_2000 | 0x4810_2FFF | 4KB | Ethos-U55 APB | Modeled |
| 22 | 0x4810_3000 | 0x4810_31FF | 0.5KB | U55 timing adapter 0 APB | Modeled |
| 23 | 0x4810_3200 | 0x4810_33FF | 0.5KB | U55 timing adapter 1 APB | Modeled |
| - | 0x4810_3400 | 0x491F_FFFF | - | Reserved | - |

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

| Row ID | Address range in non-secure region | | Size | Description | Modeled in FVP |
|---|---|---|---|---|---|
| | From | To | | | |
| 24 | 0x4920_0000 | 0x4920_0FFF | 4KB | FPGA - SBCon I2C (Touch) | Partially modeled |
| 25 | 0x4920_1000 | 0x4920_1FFF | 4KB | FPGA - SBCon I2C (Audio Conf) | Dummy stub |
| 26 | 0x4920_2000 | 0x4920_2FFF | 4KB | FPGA - PL022 (SPI ADC) | Dummy stub |
| 27 | 0x4920_3000 | 0x4920_3FFF | 4KB | FPGA - PL022 (SPI Shield0) | Dummy stub |
| 28 | 0x4920_4000 | 0x4920_4FFF | 4KB | FPGA - PL022 (SPI Shield1) | Dummy stub |
| 29 | 0x4920_5000 | 0x4920_5FFF | 4KB | SBCon (I2C - Shield0) | Dummy stub |
| 30 | 0x4920_6000 | 0x4920_6FFF | 4KB | SBCon (I2C – Shield1) | Dummy stub |
| 31 | 0x4920_7000 | 0x4920_7FFF | 4KB | USER APB | Dummy stub |
| 32 | 0x4920_8000 | 0x4920_8FFF | 4KB | FPGA - SBCon I2C (DDR4 EEPROM) | Dummy stub |
| - | 0x4920_9000 | 0x492F_FFFF | - | Reserved | - |
| 33 | 0x4930_0000 | 0x4930_0FFF | 4KB | FPGA - SCC registers | Modeled |
| 34 | 0x4930_1000 | 0x4930_1FFF | 4KB | FPGA - I2S (Audio) | Partially modeled |
| 35 | 0x4930_2000 | 0x4930_2FFF | 4KB | FPGA - IO (System Ctrl + I/O) | Modeled |
| 36 | 0x4930_3000 | 0x4930_3FFF | 4KB | UART0 - UART_F [0] | CMSDK UART |
| 37 | 0x4930_4000 | 0x4930_4FFF | 4KB | UART1 - UART_F [1] | CMSDK UART |
| 38 | 0x4930_5000 | 0x4930_5FFF | 4KB | UART2 - UART_F [2] | CMSDK UART |
| 39 | 0x4930_6000 | 0x4930_6FFF | 4KB | UART3 - UART Shield 0 | Dummy stub |
| 40 | 0x4930_7000 | 0x4930_7FFF | 4KB | UART4 - UART Shield 1 | Dummy stub |
| 41 | 0x4930_8000 | 0x4930_8FFF | 4KB | UART5 - UART_F [3] | CMSDK UART |
| 42 | 0x4930_9000 | 0x4930_9FFF | 4KB | Reserved | |
| 43 | 0x4930_A000 | 0x4930_AFFF | 4KB | CLCD Config Reg | Partially modeled |
| 44 | 0x4930_B000 | 0x4930_BFFF | 4KB | RTC | PL031_RTC |

### 5.2.2 Corstone™ SSE-300 secure expansion peripherals memory map

Memory map for secure board peripherals.

**Table 5-3: Secure board peripherals**

| Row ID | Address range in secure region | | Size | Description | Modeled in FVP |
|---|---|---|---|---|---|
| | From | To | | | |
| 1 | 0x5110_0000 | 0x5110_0FFF | 4KB | GPIO 0 | CMSDK GPIO |

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

| Row ID | Address range in secure region | | Size | Description | Modeled in FVP |
|---|---|---|---|---|---|
| | From | To | | | |
| 2 | 0x5110_1000 | 0x5110_1FFF | 4KB | GPIO 1 | CMSDK GPIO |
| 3 | 0x5110_2000 | 0x5110_2FFF | 4KB | GPIO 2 | CMSDK GPIO |
| 4 | 0x5110_3000 | 0x5110_3FFF | 4KB | GPIO 3 | CMSDK GPIO |
| 5 | 0x5110_4000 | 0x5110_4FFF | 4KB | FMC GPIO 0 | CMSDK GPIO |
| 6 | 0x5110_5000 | 0x5110_5FFF | 4KB | FMC GPIO 1 | CMSDK GPIO |
| 7 | 0x5110_6000 | 0x5110_6FFF | 4KB | FMC GPIO 2 | CMSDK GPIO |
| 8 | 0x5110_7000 | 0x5110_7FFF | 4KB | FMC USER AHB | Not modeled |
| | 0x5110_8000 | 0x511F_FFFF | | Reserved | |
| 9 | 0x5120_0000 | 0x5120_0FFF | 4KB | USER AHB 0 | Not modeled |
| 10 | 0x5120_1000 | 0x5120_1FFF | 4KB | USER AHB 1 | Not modeled |
| 11 | 0x5120_2000 | 0x5120_2FFF | 4KB | USER AHB 2 | Not modeled |
| 12 | 0x5120_3000 | 0x5120_3FFF | 4KB | USER AHB 3 | Not modeled |
| | 0x5120_4000 | 0x513F_FFFF | | Reserved | |
| 13 | 0x5140_0000 | 0x514F_FFFF | 1MB | Ethernet | SMSC91C111 Ethernet controller |
| 14 | 0x5150_0000 | 0x515F_FFFF | 1MB | USB | Dummy stub |
| | 0x5160_0000 | 0x516F_FFFF | | Reserved | |
| 15 | 0x5170_0000 | 0x5170_0FFF | 4KB | User APB0 | Not modeled |
| 16 | 0x5170_1000 | 0x5170_1FFF | 4KB | User APB1 | Not modeled |
| 17 | 0x5170_2000 | 0x5170_2FFF | 4KB | User APB2 | Not modeled |
| 18 | 0x5170_3000 | 0x5170_3FFF | 4KB | User APB3 | Not modeled |
| | 0x5170_4000 | 0x517F_FFFF | | Reserved | |
| 19 | 0x5180_0000 | 0x5180_0FFF | 4KB | QSPI Config | Not modeled |
| 20 | 0x5180_1000 | 0x5180_1FFF | 4KB | QSPI Write | Not modeled |
| | 0x5180_2000 | 0x56FF_FFFF | | Reserved | |
| 21 | 0x5700_0000 | 0x5700_0FFF | 4KB | SRAM Memory Protection Controller (MPC) | Modeled |
| 22 | 0x5700_1000 | 0x5700_1FFF | 4KB | QSPI Memory Protection Controller (MPC) | Modeled |
| 23 | 0x5700_2000 | 0x5700_2FFF | 4KB | DDR4 Memory Protection Controller (MPC) | Modeled |
| | 0x5700_3000 | 0x57FF_FFFF | | Reserved | |
| | 0x5800_0000 | 0x580F_FFFF | | Subsystem peripherals | |
| 24 | 0x5810_2000 | 0x5810_2FFF | 4KB | Ethos-U55 APB | Modeled |
| 25 | 0x5810_3000 | 0x5810_31FF | 0.5KB | U55 timing adapter 0 APB | Modeled |

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

| Row ID | Address range in secure region | | Size | Description | Modeled in FVP |
|---|---|---|---|---|---|
| | From | To | | | |
| 26 | 0x5810_3200 | 0x5810_33FF | 0.5KB | U55 timing adapter 1 APB | Modeled |
| | 0x5810_3400 | 0x591F_FFFF | | Reserved | |
| 27 | 0x5920_4000 | 0x5920_4FFF | 4KB | FPGA - PL022 (SPI Shield1) | Dummy stub |
| 28 | 0x5920_5000 | 0x5920_5FFF | 4KB | SBCon (I2C - Shield0) | Dummy stub |
| 29 | 0x5920_6000 | 0x5920_6FFF | 4KB | SBCon (I2C – Shield1) | Dummy stub |
| 30 | 0x5920_7000 | 0x5920_7FFF | 4KB | USER APB | Dummy stub |
| 31 | 0x5920_8000 | 0x5920_8FFF | 4KB | DDR4 EEPROM | Dummy stub |
| | 0x5920_9000 | 0x592F_FFFF | | Reserved | |
| 32 | 0x5930_0000 | 0x5930_0FFF | 4KB | FPGA - SCC registers | Modeled |
| 33 | 0x5930_1000 | 0x5930_1FFF | 4KB | FPGA - I2S (Audio) | Partially modeled |
| 34 | 0x5930_2000 | 0x5930_2FFF | 4KB | FPGA - IO (System Ctrl + I/O) | Modeled |
| 35 | 0x5930_3000 | 0x5930_3FFF | 4KB | UART0 - UART_F [0] | CMSDK UART |
| 36 | 0x5930_4000 | 0x5930_4FFF | 4KB | UART1 - UART_F [1] | CMSDK UART |
| 37 | 0x5930_5000 | 0x5930_5FFF | 4KB | UART2 - UART_F [2] | CMSDK UART |
| 38 | 0x5930_6000 | 0x5930_6FFF | 4KB | UART3 - UART Shield 0 | Dummy stub |
| 39 | 0x5930_7000 | 0x5930_7FFF | 4KB | UART4 - UART Shield 1 | Dummy stub |
| 40 | 0x5930_8000 | 0x5930_8FFF | 4KB | UART5 - UART_F [3] | CMSDK UART |
| | 0x5930_9000 | 0x5930_9FFF | 4KB | Reserved | |
| 41 | 0x5930_A000 | 0x5930_AFFF | 4KB | CLCD Config Reg | Partially modeled |
| 42 | 0x5930_B000 | 0x5930_BFFF | 4KB | RTC | PL031_RTC |

### 5.2.3 Corstone™ SSE-300 expansion peripheral interrupt map

Interrupt map at the board layer.

**Table 5-4: Interrupt map at the board layer**

| Interrupt input | Interrupt source |
|---|---|
| IRQ [32] | System timestamp counter interrupt. Not implemented. |
| IRQ [33] | UART 0 receive interrupt. |
| IRQ [34] | UART 0 transmit Interrupt. |
| IRQ [35] | UART 1 receive interrupt. |
| IRQ [36] | UART 1 transmit interrupt. |
| IRQ [37] | UART 2 receive interrupt. |
| IRQ [38] | UART 2 transmit interrupt. |

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

| Interrupt input | Interrupt source |
|---|---|
| IRQ [39] | UART 3 receive interrupt. |
| IRQ [40] | UART 3 transmit interrupt. |
| IRQ [41] | UART 4 receive interrupt. |
| IRQ [42] | UART 4 transmit interrupt. |
| IRQ [43] | UART 0 combined interrupt. |
| IRQ [44] | UART 1 combined interrupt. |
| IRQ [45] | UART 2 combined interrupt. |
| IRQ [46] | UART 3 combined interrupt. |
| IRQ [47] | UART 4 combined interrupt. |
| IRQ [48] | UART overflow (0, 1, 2, 3, 4, and 5). |
| IRQ [49] | Ethernet. |
| IRQ [50] | Audio I2S. |
| IRQ [51] | Touch screen. |
| IRQ [52] | USB. |
| IRQ [53] | SPI ADC. |
| IRQ [54] | SPI (shield 0). |
| IRQ [55] | SPI (shield 1). |
| IRQ [56] | Ethos-U interrupt. |
| IRQ [68:57] | Reserved. |
| IRQ [69] | GPIO 0 combined interrupt. |
| IRQ [70] | GPIO 1 combined interrupt. |
| IRQ [71] | GPIO 2 combined interrupt. |
| IRQ [72] | GPIO 3 combined interrupt. |
| IRQ [88:73] | GPIO 0 individual interrupts. |
| IRQ [104:89] | GPIO 1 individual interrupts. |
| IRQ [120:105] | GPIO 2 individual interrupts. |
| IRQ [124:121] | GPIO 3 individual interrupts. |
| IRQ [125] | UART 5 receive interrupt. |
| IRQ [126] | UART 5 transmit interrupt. |
| IRQ [127] | UART 5 combined interrupt. |
| IRQ [130:128] | Reserved. |

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

## 5.3 Corstone™ SSE-300 peripheral protection controller expansion map

The Corstone™ SSE-300 FVP implements secure access configuration registers which control security and privileged accesses to peripherals connected to PPC.

**Table 5-5: Secure access configuration registers - PPC bits**

| Bit | MAIN_PPCEXP0 (AHB0) | MAIN_PPCEXP1 (AHB1) | PERIPH_PPCEXP0 (APB0) | PERIPH_PPCEXP1 (APB1) | PERIPH_PPCEXP2 (APB2) |
|---|---|---|---|---|---|
| 0 | GPIO-0 | User AHB 0 | - | SBCon I2C (touch screen) | FPGA - SCC registers |
| 1 | GPIO-1 | User AHB 1 | - | SBCon I2C (audio conf) | FPGA - I2S (audio) |
| 2 | GPIO-2 | User AHB 2 | - | FPGA PL022 (SPI2 for ADC) | FPGA – GPIO (System Ctrl + I/O) |
| 3 | GPIO-3 | User AHB 3 | - | FPGA PL022 (SPI Shield0) | UART0 |
| 4 | FMC GPIO-0 | - | NPU APB0 | FPGA PL022 (SPI Shield1) | UART1 |
| 5 | FMC GPIO-1 | - | NPU APB1 | FPGA SBCon (I2C - Shield0) | UART2 |
| 6 | FMC GPIO-2 | - | - | FPGA SBCon (I2C – Shield1) | UART3 STUB |
| 7 | FMC User AHB | - | - | Reserved | UART4 STUB |
| 8 | USB and Ethernet | - | - | FPGA – SBCon I2C (DDR4 EPROM) | UART5 |
| 9 | - | - | - | - | Reserved |
| 10 | - | - | - | - | CLCD config reg |
| 11 | - | - | - | - | RTC |
| 12 | - | - | - | - | - |
| 13 | - | - | SRAM MPC | - | - |
| 14 | - | - | QSPI MPC | - | - |
| 15 | - | - | DDR MPC | - | - |
| PPC IRQ no. | 5 | 6 | 2 | 3 | 4 |

## 5.4 Corstone™ SSE-300 memory components

The Corstone™ SSE-300 FVP includes the following memory components and their security access is controlled by the MPC.

**Table 5-6: Memory components**

| Name | Non-secure address range | Secure alias | Size |
|---|---|---|---|
| SRAM | `0x0100_0000 - 0x010F_FFFF` | `0x1100_0000 - 0x110F_FFFF` | 1MB |
| QSPI SRAM | `0x2800_0000 - 0x287F_FFFF` | `0x3800_0000 - 0x387F_FFFF` | 8MB |
| DDR0 | `0x6000_0000 - 0x6FFF_FFFF` | `0x7000_0000 - 0x7FFF_FFFF` | 256MB |
| DDR1 | `0x8000_0000 - 0x8FFF_FFFF` | `0x9000_0000 - 0x9FFF_FFFF` | 256MB |

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

| Name | Non-secure address range | Secure alias | Size |
|---|---|---|---|
| DDR2 | `0xA000_0000 - 0xAFFF_FFFF` | `0xB000_0000 - 0xBFFF_FFFF` | 256MB |
| DDR3 | `0xC000_0000 - 0xCFFF_FFFF` | `0xD000_0000 - 0xDFFF_FFFF` | 256MB |

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

# 6. Arm® Corstone™ SSE-310 FVP

To develop ahead of hardware availability and explore the design from a software perspective, the Corstone™ SSE-310 Fixed Virtual Platform (FVP) models much of the Arm® IP in the Corstone™ SSE-310 design. It models version r0p0 of the Corstone™ SSE-310 subsystem.

The FVP drives system architecture and software standardization. It is used with the Corstone™ SSE-310 software package which provides software and binaries of proprietary firmware that reduce the amount of work that is required for SoC development. See the Corstone™ SSE-310 software bundle readme for instructions on how to set up and run the FVP.

The FVP models the following IP components:

- A single Arm® Cortex®-M85 processor with MVE extension.
- A single Arm® Ethos™-U55 processor.
- Memory Protection Controller (MPC).
- Peripheral Protection Controller (PPC).
- Implementation Defined Attribution Unit (IDAU).

The full description of components that are internal to the SSE-310 subsystem can be found in the SSE-310 Example Subsystem TRM.

The FVP has the following limitations:

- It does not model every component that Corstone™ SSE-310 describes. For example, it does not model the CoreSight™ technology components.
- QSPI is not modeled. It has QSPI SRAM instead.
- It has a single DMA-350 engine instead of four PL081 DMA peripherals.
- Some components are dummy stubs with a minimal implementation. Models with dummy APB have memory access.
- Some components are partially modeled.

Refer to the following documents for more information:

- Arm Corstone SSE-310 Example Subsystem Technical Reference Manual
- Arm MPS3 FPGA Prototyping Board Technical Reference Manual
- Corstone-310 on Arm Developer

## 6.1 Corstone™ SSE-310 FVP memory map overview

This table outlines the main FVP memories and their positions within the memory map.

This memory map includes IDAU security information for memory regions:

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**Table 6-1: Memory map overview**

| Row ID | Address range | | Size | Description | Alias with Row ID | IDAU region values | | |
|---|---|---|---|---|---|---|---|---|
| | From | To | | | | Security | IDAUID | NSC |
| 1 | 0x0000_0000 | 0x0000_7FFF | 32KB | ITCM[6] | 5 | NS | 0 | 0 |
| 2 | 0x0000_8000 | 0x00FF_FFFF | 15.9MB | Reserved | - | | | |
| 3 | 0x0100_0000 | 0x011F_FFFF | 2MB | SRAM (2MB)[4] | 7 | | | |
| 4 | 0x0120_0000 | 0x0FFF_FFFF | 238MB | Reserved | - | | | |
| 5 | 0x1000_0000 | 0x1000_7FFF | 32KB | ITCM[6] | 1 | S | 1 | CODE NSC |
| 6 | 0x1000_8000 | 0x10FF_FFFF | 15.9MB | Reserved | - | | | |
| 7 | 0x1100_0000 | 0x111F_FFFF | 2MB | SRAM (2MB)[4] | 3 | | | |
| 8 | 0x1120_0000 | 0x1FFF_FFFF | 238MB | Reserved | - | | | |
| 9 | 0x2000_0000 | 0x2000_7FFF | 32KB | DTCM (4 x banks of 8KB)[6] | 15 | NS | 2 | 0 |
| 10 | 0x2000_8000 | 0x20FF_FFFF | 15.9MB | Reserved | - | - | - | - |
| 11 | 0x2100_0000 | 0x213F_FFFF | 4MB | Internal SRAM area (SSE-310 implements 2x2MB)[6] | 17 | - | - | - |
| 12 | 0x2140_0000 | 0x27FF_FFFF | 108MB | Reserved | - | - | - | - |
| 13 | 0x2800_0000 | 0x287F_FFFF | 8MB | QSPI (only 8MB)[4] | 19 | - | - | - |
| 14 | 0x2880_0000 | 0x2FFF_FFFF | 120MB | Reserved | - | - | - | - |
| 15 | 0x3000_0000 | 0x3000_7FFF | 32KB | DTCM (4 x banks of 8KB)[6] | 9 | S | 3 | RAM NSC |
| 16 | 0x3000_8000 | 0x30FF_FFFF | 15.9MB | Reserved | - | - | - | - |
| 17 | 0x3100_0000 | 0x313F_FFFF | 4MB | Internal SRAM area (SSE-310 implements 2x2MB)[6] | 11 | - | - | - |
| 18 | 0x3140_0000 | 0x37FF_FFFF | 108MB | Reserved | - | - | - | - |
| 19 | 0x3800_0000 | 0x387F_FFFF | 8MB | QSPI (only 8MB)[4] | 13 | - | - | - |
| 20 | 0x3880_0000 | 0x3FFF_FFFF | 120MB | Reserved | - | - | - | - |
| 21 | 0x4000_0000 | 0x47FF_FFFF | 128MB | Non-Secure Low Latency Peripheral Region. | 23 | NS | 4 | 0 |
| 22 | 0x4800_0000 | 0x4FFF_FFFF | 128MB | Non-Secure High Latency Peripheral Region. | 24 | NS | 4 | 0 |
| 23 | 0x5000_0000 | 0x57FF_FFFF | 128MB | Secure Low Latency Peripheral Region. | 21 | S | 5 | 0 |
| 24 | 0x5800_0000 | 0x5FFF_FFFF | 128MB | Secure High Latency Peripheral Region. | 22 | S | 5 | 0 |
| 25 | 0x6000_0000 | 0x6FFF_FFFF | 256MB | DDR4[4] | - | NS | 6 | 0 |
| 26 | 0x7000_0000 | 0x7FFF_FFFF | 256MB | DDR4[4] | - | S | 7 | 0 |
| 27 | 0x8000_0000 | 0x8FFF_FFFF | 256MB | DDR4[4] | - | NS | 8 | 0 |
| 28 | 0x9000_0000 | 0x9FFF_FFFF | 256MB | DDR4[4] | - | S | 9 | 0 |
| 29 | 0xA000_0000 | 0xAFFF_FFFF | 256MB | DDR4[4] | - | NS | A | 0 |
| 30 | 0xB000_0000 | 0xBFFF_FFFF | 256MB | DDR4[4] | - | S | B | 0 |
| 31 | 0xC000_0000 | 0xCFFF_FFFF | 256MB | DDR4[4] | - | NS | C | 0 |

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

| Row ID | Address range | | Size | Description | Alias with Row ID | IDAU region values | | |
|---|---|---|---|---|---|---|---|---|
| | From | To | | | | Security | IDAUID | NSC |
| 32 | `0xD000_0000` | `0xDFFF_FFFF` | 256MB | DDR4[4] | - | S | D | 0 |
| 33 | `0xE000_0000` | `0xE00F_FFFF` | 1MB | External Private Peripheral Bus | - | - | Exempt | - |
| 34 | `0xE010_0000` | `0xE01F_FFFF` | 1MB | Reserved | - | NS | E | 0 |
| 35 | `0xE020_0000` | `0xEFFF_FFFF` | 254MB | Maps to HMSTEXPPILL expansion interface[5] | - | NS | E | 0 |
| 36 | `0xF000_0000` | `0xF00F_FFFF` | 1MB | Reserved | - | - | Exempt | - |
| 37 | `0xF010_0000` | `0xF01F_FFFF` | 1MB | Reserved | - | S | F | 0 |
| 38 | `0xF020_0000` | `0xFFFF_FFFF` | 254MB | Maps to HMSTEXPPILL expansion interface[5] | - | S | F | 0 |

## 6.2 Corstone™ SSE-310 FVP peripherals

The Corstone™ SSE-310 FVP includes peripherals that the software payload requires to run.

These peripherals are organized in the following layers:

**Subsystem**

The subsystem peripherals represent peripherals that are present on the SoC.

**Board**

The board peripherals represent peripherals that may be present on the board onto which the SoC is mounted. The Corstone™ SSE-310 board model is based on the Arm® MPS3 board.

All peripherals that are extensions to the Corstone™ SSE-310 subsystem are mapped into two key areas of the memory map:

**`0x4000_0000` to `0x47FF_FFFF` and `0x4800_0000` to `0x4FFF_FFFF`**

Non-Secure region that maps to the AHB Master Expansion 1 interface.

**`0x5000_0000` to `0x57FF_FFFF` and `0x5800_0000` to `0x5FFF_FFFF`**

Secure region that maps to the AHB Master Expansion 1 interface.

---

Note

Some of the MPS3 Fast Models have minimal implementations. For more information, refer to the documents in the FVP package.

---

[4] Security access is controlled by MPC.
[5] Accesses to these addresses result in an AHB5 error response.
[6] For security settings, control, and features, refer to the Arm® Corstone™ SSE-310 documentation.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

### 6.2.1 Corstone™ SSE-310 FVP subsystem peripherals non-secure memory map

Non-secure memory map for subsystem peripherals.

**Table 6-2: Non-secure subsystem peripherals**

| Row ID | Address range in non-secure region | | Size | Description | Modeled in FVP |
|---|---|---|---|---|---|
| | From | To | | | |
| 1 | 0x4000_0000 | 0x4000_0FFF | 4KB | MHU 0 | Not modeled |
| 2 | 0x4000_1000 | 0x4000_1FFF | 4KB | MHU 1 | Not modeled |
| 3 | 0x4000_2000 | 0x4000_3FFF | 8KB | DMA | DMA-350 |
| 4 | 0x4000_4000 | 0x4000_4FFF | 4KB | NPU APB | Modeled |
| - | 0x4000_5000 | 0x4001_EFFF | - | Reserved | - |
| 5 | 0x4001_F000 | 0x4001_FFFF | 4KB | CPUID | Modeled |
| - | 0x4002_0000 | 0x4007_0000 | - | Reserved | - |
| 6 | 0x4008_0000 | 0x4008_0FFF | 4KB | Non-secure Access Configuration Registers | Modeled |
| - | 0x4008_1000 | 0x4008_FFFF | - | Reserved | - |
| 7 | 0x4009_0000 | 0x4009_3FFF | 16KB | CryptoCell312 | Not modeled |
| - | 0x4009_4000 | 0x400F_FFFF | - | Reserved | - |
| - | 0x4010_0000 | 0x47FF_FFFF | - | Board peripherals | - |
| 8 | 0x4800_0000 | 0x4800_0FFF | 4KB | Timer 0 | Modeled |
| 9 | 0x4800_1000 | 0x4800_1FFF | 4KB | Timer 1 | Modeled |
| 10 | 0x4800_2000 | 0x4800_2FFF | 4KB | Timer 2 | Modeled |
| 11 | 0x4800_3000 | 0x4800_3FFF | 4KB | Timer 3 | Modeled |
| - | 0x4800_4000 | 0x4801_FFFF | - | Reserved | - |
| 12 | 0x4802_0000 | 0x4802_0FFF | 4KB | SYSINFO | Modeled |
| - | 0x4802_1FFF | 0x4802_EFFF | - | Reserved | - |
| 13 | 0x4802_F000 | 0x4802_FFFF | 4KB | SLOWCLK Timer | Modeled |
| - | 0x4803_0000 | 0x4803_FFFF | - | Reserved | - |
| 14 | 0x4804_0000 | 0x4804_0FFF | 4KB | Non-Secure Watchdog Control Frame | Modeled |
| 15 | 0x4804_1000 | 0x4804_1FFF | 4KB | Non-Secure Watchdog Refresh Frame | Modeled |
| - | 0x4804_2000 | 0x480F_FFFF | - | Reserved | - |

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

### 6.2.2 Corstone™ SSE-310 FVP subsystem peripherals secure memory map

Secure memory map for subsystem peripherals.

**Table 6-3: Secure subsystem peripherals**

| Row ID | Address range in non-secure region | | Size | Description | Modeled in FVP |
|---|---|---|---|---|---|
| | From | To | | | |
| 1 | 0x5000_0000 | 0x5000_0FFF | 4KB | MHU 0 | Not modeled |
| 2 | 0x5000_1000 | 0x5000_1FFF | 4KB | MHU 1 | Not modeled |
| 3 | 0x5000_2000 | 0x5000_3FFF | 8KB | DMA | DMA-350 |
| 4 | 0x5000_4000 | 0x5000_4FFF | 4KB | NPU APB | Modeled |
| - | 0x5000_5000 | 0x5001_EFFF | - | Reserved | - |
| 5 | 0x5001_F000 | 0x5001_FFFF | 4KB | CPUID | Modeled |
| - | 0x5002_0000 | 0x5007_0000 | - | Reserved | - |
| 6 | 0x5008_0000 | 0x5008_0FFF | 4KB | Secure Access Configuration Registers | Modeled |
| - | 0x5008_1000 | 0x5008_2FFF | - | Reserved | - |
| 7 | 0x5008_3000 | 0x5008_3FFF | 4KB | Internal SRAM MPC 0 | Modeled |
| 8 | 0x5008_4000 | 0x5008_4FFF | 4KB | Internal SRAM MPC 1 | Modeled |
| - | 0x5008_5000 | 0x5008_FFFF | - | Reserved | - |
| 9 | 0x5009_0000 | 0x5009_3FFF | 16KB | CryptoCell312 | Not modeled |
| - | 0x5009_4000 | 0x500F_FFFF | - | Reserved | - |
| - | 0x5010_0000 | 0x57FF_FFFF | - | Board peripherals | - |
| 10 | 0x5800_0000 | 0x5800_0FFF | 4KB | Timer 0 | Modeled |
| 11 | 0x5800_1000 | 0x5800_1FFF | 4KB | Timer 1 | Modeled |
| 12 | 0x5800_2000 | 0x5800_2FFF | 4KB | Timer 2 | Modeled |
| 13 | 0x5800_3000 | 0x5800_3FFF | 4KB | Timer 3 | Modeled |
| - | 0x5800_4000 | 0x5801_FFFF | - | Reserved | - |
| 14 | 0x5802_0000 | 0x5802_0FFF | 4KB | SYSINFO | Modeled |
| 15 | 0x5802_1000 | 0x5802_1FFF | 4KB | SYSCONTROL | Modeled |
| 16 | 0x5802_2000 | 0x5802_2FFF | 4KB | SYS_PPU | Modeled |
| 17 | 0x5802_3000 | 0x5802_3FFF | 4KB | CPU0_PPU | Modeled |
| - | 0x5802_4000 | 0x5802_6FFF | - | Reserved | - |
| 18 | 0x5802_7000 | 0x5802_7FFF | 4KB | CRYPTO_PPU | Not modeled |
| 19 | 0x5802_8000 | 0x5802_8FFF | 4KB | MGMT_PPU | Modeled |
| 20 | 0x5802_9000 | 0x5802_9FFF | 4KB | DBG_PPU | Modeled |
| 21 | 0x5802_A000 | 0x5802_AFFF | 4KB | NPU_PPU | Modeled |
| - | 0x5802_B000 | 0x5802_DFFF | - | Reserved | - |
| 22 | 0x5802_E000 | 0x5802_EFFF | 4KB | SLOWCLK Watchdog | Modeled |

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

| Row ID | Address range in non-secure region | | Size | Description | Modeled in FVP |
|---|---|---|---|---|---|
| | From | To | | | |
| 23 | 0x5802_F000 | 0x5802_FFFF | 4KB | SLOWCLK Timer | Modeled |
| - | 0x5803_0000 | 0x5803_FFFF | - | Reserved | - |
| 24 | 0x5804_0000 | 0x5804_0FFF | 4KB | Secure Watchdog Control Frame | Modeled |
| 25 | 0x5804_1000 | 0x5804_1FFF | 4KB | Secure Watchdog Refresh Frame | Modeled |
| - | 0x5804_2000 | 0x580F_FFFF | - | Reserved | - |

### 6.2.3 Corstone™ SSE-310 FVP non-secure expansion peripherals memory map

Non-secure memory map for board peripherals.

**Table 6-4: Non-secure board peripherals**

| Row ID | Address range in non-secure region | | Size | Description | Modeled in FVP |
|---|---|---|---|---|---|
| | From | To | | | |
| 1 | 0x4110_0000 | 0x4110_0FFF | 4KB | GPIO 0 | CMSDK GPIO |
| 2 | 0x4110_1000 | 0x4110_1FFF | 4KB | GPIO 1 | CMSDK GPIO |
| 3 | 0x4110_2000 | 0x4110_2FFF | 4KB | GPIO 2 | CMSDK GPIO |
| 4 | 0x4110_3000 | 0x4110_3FFF | 4KB | GPIO 3 | CMSDK GPIO |
| 5 | 0x4110_4000 | 0x4110_4FFF | 4KB | USER AHB 0 | Not modeled |
| 6 | 0x4110_5000 | 0x4110_5FFF | 4KB | USER AHB 1 | Not modeled |
| 7 | 0x4110_6000 | 0x4110_6FFF | 4KB | USER AHB 2 | Not modeled |
| 8 | 0x4110_7000 | 0x4110_7FFF | 4KB | USER AHB 3 | Not modeled |
| - | 0x4110_8000 | 0x413F_FFFF | - | Reserved | - |
| 9 | 0x4140_0000 | 0x414F_FFFF | 1MB | Ethernet | SMSC91C111 Ethernet controller |
| 10 | 0x4150_0000 | 0x415F_FFFF | 1MB | USB | Dummy stub |
| - | 0x4160_0000 | 0x416F_FFFF | - | Reserved | - |
| 11 | 0x4170_0000 | 0x4170_0FFF | 4KB | Timing adapter APB0 - FPGA SRAM | Dummy stub |
| 12 | 0x4170_1000 | 0x4170_1FFF | 4KB | Timing adapter APB1 - QSPI | Dummy stub |
| 13 | 0x4170_2000 | 0x4170_2FFF | 4KB | Timing adapter APB2 - DDR4 | Dummy stub |
| 14 | 0x4170_3000 | 0x4170_3FFF | 4KB | User APB3 | Not modeled |
| - | 0x4170_4000 | 0x417F_FFFF | - | Reserved | - |
| 15 | 0x4180_0000 | 0x4180_0FFF | 4KB | QSPI Config | Not modeled |
| 16 | 0x4180_1000 | 0x4180_1FFF | 4KB | QSPI Write | Not modeled |
| - | 0x4180_2000 | 0x47FF_FFFF | - | Reserved | - |

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

| Row ID | Address range in non-secure region | | Size | Description | Modeled in FVP |
|---|---|---|---|---|---|
| | From | To | | | |
| - | 0x4800_0000 | 0x480F_FFFF | - | Subsystem peripherals | - |
| - | 0x4810_0000 | 0x491F_FFFF | - | Reserved | - |
| 17 | 0x4920_0000 | 0x4920_0FFF | 4KB | FPGA - SBCon I2C (Touch) | Partially modeled |
| 18 | 0x4920_1000 | 0x4920_1FFF | 4KB | FPGA - SBCon I2C (Audio Conf) | Dummy stub |
| 19 | 0x4920_2000 | 0x4920_2FFF | 4KB | FPGA - PL022 (SPI ADC) | Dummy stub |
| 20 | 0x4920_3000 | 0x4920_3FFF | 4KB | FPGA - PL022 (SPI Shield0) | Dummy stub |
| 21 | 0x4920_4000 | 0x4920_4FFF | 4KB | FPGA - PL022 (SPI Shield1) | Dummy stub |
| 22 | 0x4920_5000 | 0x4920_5FFF | 4KB | SBCon (I2C - Shield0) | Dummy stub |
| 23 | 0x4920_6000 | 0x4920_6FFF | 4KB | SBCon (I2C – Shield1) | Dummy stub |
| 24 | 0x4920_7000 | 0x4920_7FFF | 4KB | USER APB | Dummy stub |
| 25 | 0x4920_8000 | 0x4920_8FFF | 4KB | FPGA - SBCon I2C (DDR4 EEPROM) | Dummy stub |
| - | 0x4920_9000 | 0x492F_FFFF | - | Reserved | - |
| 26 | 0x4930_0000 | 0x4930_0FFF | 4KB | FPGA - SCC registers | Modeled |
| 27 | 0x4930_1000 | 0x4930_1FFF | 4KB | FPGA - I2S (Audio) | Partially modeled |
| 28 | 0x4930_2000 | 0x4930_2FFF | 4KB | FPGA - IO (System Ctrl + I/O) | Modeled |
| 29 | 0x4930_3000 | 0x4930_3FFF | 4KB | UART0 - UART_F [0] | CMSDK UART |
| 30 | 0x4930_4000 | 0x4930_4FFF | 4KB | UART1 - UART_F [1] | CMSDK UART |
| 31 | 0x4930_5000 | 0x4930_5FFF | 4KB | UART2 - UART_F [2] | CMSDK UART |
| 32 | 0x4930_6000 | 0x4930_6FFF | 4KB | UART3 - UART Shield 0 | Dummy stub |
| 33 | 0x4930_7000 | 0x4930_7FFF | 4KB | UART4 - UART Shield 1 | Dummy stub |
| 34 | 0x4930_8000 | 0x4930_8FFF | 4KB | UART5 - UART_F [3] | CMSDK UART |
| - | 0x4930_9000 | 0x4930_9FFF | 4KB | Reserved | - |
| 35 | 0x4930_A000 | 0x4930_AFFF | 4KB | CLCD Config Reg | Partially modeled |
| 36 | 0x4930_B000 | 0x4930_BFFF | 4KB | RTC | PL031_RTC |

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

### 6.2.4 Corstone™ SSE-310 FVP secure expansion peripherals memory map

Secure memory map for board peripherals.

**Table 6-5: Secure board peripherals**

| Row ID | Address range in secure region | | Size | Description | Modeled in FVP |
|---|---|---|---|---|---|
| | From | To | | | |
| 1 | 0x5110_0000 | 0x5110_0FFF | 4KB | GPIO 0 | CMSDK GPIO |
| 2 | 0x5110_1000 | 0x5110_1FFF | 4KB | GPIO 1 | CMSDK GPIO |
| 3 | 0x5110_2000 | 0x5110_2FFF | 4KB | GPIO 2 | CMSDK GPIO |
| 4 | 0x5110_3000 | 0x5110_3FFF | 4KB | GPIO 3 | CMSDK GPIO |
| 5 | 0x5110_4000 | 0x5110_4FFF | 4KB | USER AHB 0 | Not modeled |
| 6 | 0x5110_5000 | 0x5110_5FFF | 4KB | USER AHB 1 | Not modeled |
| 7 | 0x5110_6000 | 0x5110_6FFF | 4KB | USER AHB 2 | Not modeled |
| 8 | 0x5110_7000 | 0x5110_7FFF | 4KB | USER AHB 3 | Not modeled |
| - | 0x5110_8000 | 0x513F_FFFF | - | Reserved | - |
| 9 | 0x5140_0000 | 0x514F_FFFF | 1MB | Ethernet | SMSC91C111 Ethernet controller |
| 10 | 0x5150_0000 | 0x515F_FFFF | 1MB | USB | Dummy stub |
| - | 0x5160_0000 | 0x516F_FFFF | - | Reserved | - |
| 11 | 0x5170_0000 | 0x5170_0FFF | 4KB | Timing Adapter APB0 - FPGA SRAM | Dummy stub |
| 12 | 0x5170_1000 | 0x5170_1FFF | 4KB | Timing Adapter APB1 - QSPI | Dummy stub |
| 13 | 0x5170_2000 | 0x5170_2FFF | 4KB | Timing Adapter APB2 - DDR4 | Dummy stub |
| 14 | 0x5170_3000 | 0x5170_3FFF | 4KB | User APB3 | Not modeled |
| - | 0x5170_4000 | 0x517F_FFFF | - | Reserved | - |
| 15 | 0x5180_0000 | 0x5180_0FFF | 4KB | QSPI Config | Not modeled |
| 16 | 0x5180_1000 | 0x5180_1FFF | 4KB | QSPI Write | Not modeled |
| - | 0x5180_2000 | 0x56FF_FFFF | - | Reserved | - |
| 17 | 0x5700_0000 | 0x5700_0FFF | 4KB | SRAM Memory Protection Controller (MPC) | Modeled |
| 18 | 0x5700_1000 | 0x5700_1FFF | 4KB | QSPI Memory Protection Controller (MPC) | Modeled |
| 19 | 0x5700_2000 | 0x5700_2FFF | 4KB | DDR4 Memory Protection Controller (MPC) | Modeled |
| - | 0x5700_3000 | 0x57FF_FFFF | - | Reserved | - |
| - | 0x5800_0000 | 0x5810_1FFF | - | Subsystem peripherals | - |
| - | 0x5810_2000 | 0x591F_FFFF | - | Reserved | - |

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

| Row ID | Address range in secure region | | Size | Description | Modeled in FVP |
|---|---|---|---|---|---|
| | From | To | | | |
| 20 | 0x5920_0000 | 0x5920_0FFF | 4KB | FPGA - SBCon I2C (Touch) | Partially modeled |
| 21 | 0x5920_1000 | 0x5920_1FFF | 4KB | FPGA - SBCon I2C (Audio Conf) | Dummy stub |
| 22 | 0x5920_2000 | 0x5920_2FFF | 4KB | FPGA - PL022 (SPI ADC) | Dummy stub |
| 23 | 0x5920_3000 | 0x5920_3FFF | 4KB | FPGA - PL022 (SPI Shield0) | Dummy stub |
| 24 | 0x5920_4000 | 0x5920_4FFF | 4KB | FPGA - PL022 (SPI Shield1) | Dummy stub |
| 25 | 0x5920_5000 | 0x5920_5FFF | 4KB | SBCon (I2C - Shield0) | Dummy stub |
| 26 | 0x5920_6000 | 0x5920_6FFF | 4KB | SBCon (I2C – Shield1) | Dummy stub |
| 27 | 0x5920_7000 | 0x5920_7FFF | 4KB | USER APB | Dummy stub |
| 28 | 0x5920_8000 | 0x5920_8FFF | 4KB | DDR4 EEPROM | Dummy stub |
| - | 0x5920_9000 | 0x592F_FFFF | - | Reserved | - |
| 29 | 0x5930_0000 | 0x5930_0FFF | 4KB | FPGA - SCC registers | Modeled |
| 30 | 0x5930_1000 | 0x5930_1FFF | 4KB | FPGA - I2S (Audio) | Partially modeled |
| 31 | 0x5930_2000 | 0x5930_2FFF | 4KB | FPGA - IO (System Ctrl + I/O) | Modeled |
| 32 | 0x5930_3000 | 0x5930_3FFF | 4KB | UART0 - UART_F [0] | CMSDK UART |
| 33 | 0x5930_4000 | 0x5930_4FFF | 4KB | UART1 - UART_F [1] | CMSDK UART |
| 34 | 0x5930_5000 | 0x5930_5FFF | 4KB | UART2 - UART_F [2] | CMSDK UART |
| 35 | 0x5930_6000 | 0x5930_6FFF | 4KB | UART3 - UART Shield 0 | Dummy stub |
| 36 | 0x5930_7000 | 0x5930_7FFF | 4KB | UART4 - UART Shield 1 | Dummy stub |
| 37 | 0x5930_8000 | 0x5930_8FFF | 4KB | UART5 - UART_F [3] | CMSDK UART |
| - | 0x5930_9000 | 0x5930_9FFF | 4KB | Reserved | - |
| 38 | 0x5930_A000 | 0x5930_AFFF | 4KB | CLCD Config Reg | Partially modeled |
| 39 | 0x5930_B000 | 0x5930_BFFF | 4KB | RTC | PL031_RTC |

### 6.2.5 Corstone SSE-310 FVP expansion peripheral interrupt map

Interrupt map at the board layer.

**Table 6-6: Interrupt map at the board layer**

| Interrupt input | Interrupt source |
|---|---|
| IRQ [32] | System timestamp counter interrupt. Not implemented. |
| IRQ [33] | UART 0 receive interrupt. |
| IRQ [34] | UART 0 transmit interrupt. |

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

| Interrupt input | Interrupt source |
|---|---|
| IRQ [35] | UART 1 receive interrupt. |
| IRQ [36] | UART 1 transmit interrupt. |
| IRQ [37] | UART 2 receive interrupt. |
| IRQ [38] | UART 2 transmit interrupt. |
| IRQ [39] | UART 3 receive interrupt. |
| IRQ [40] | UART 3 transmit interrupt. |
| IRQ [41] | UART 4 receive interrupt. |
| IRQ [42] | UART 4 transmit interrupt. |
| IRQ [43] | UART 0 combined interrupt. |
| IRQ [44] | UART 1 combined interrupt. |
| IRQ [45] | UART 2 combined interrupt. |
| IRQ [46] | UART 3 combined interrupt. |
| IRQ [47] | UART 4 combined interrupt. |
| IRQ [48] | UART overflow (0, 1, 2, 3, 4, and 5). |
| IRQ [49] | Ethernet. |
| IRQ [50] | Audio I2S. |
| IRQ [51] | Touch screen. |
| IRQ [52] | USB. |
| IRQ [53] | SPI ADC. |
| IRQ [54] | SPI (shield 0). |
| IRQ [55] | SPI (shield 1). |
| IRQ [56] | Reserved. |
| IRQ [57] | DMA channel 0 interrupt. |
| IRQ [58] | DMA channel 1 interrupt. |
| IRQ [68:59] | Reserved. |
| IRQ [69] | GPIO 0 combined interrupt. |
| IRQ [70] | GPIO 1 combined interrupt. |
| IRQ [71] | GPIO 2 combined interrupt. |
| IRQ [72] | GPIO 3 combined interrupt. |
| IRQ [88:73] | GPIO 0 individual interrupts. |
| IRQ [104:89] | GPIO 1 individual interrupts. |
| IRQ [120:105] | GPIO 2 individual interrupts. |
| IRQ [124:121] | GPIO 3 individual interrupts. |
| IRQ [125] | UART 5 receive interrupt. |
| IRQ [126] | UART 5 transmit interrupt. |
| IRQ [127] | UART 5 combined interrupt. |
| IRQ [130:128] | Reserved. |

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

## 6.3 Corstone™ SSE-310 FVP peripheral protection controller expansion map

The Corstone™ SSE-310 FVP implements secure access configuration registers which control security and privileged accesses to peripherals connected to PPC.

**Table 6-7: Secure access configuration registers - PPC bits**

| Bit | MAIN_PPCEXP0 (AHB0) | MAIN_PPCEXP1 (AHB1) | PERIPH_PPCEXP0 (APB0) | PERIPH_PPCEXP1 (APB1) | PERIPH_PPCEXP2 (APB2) |
|---|---|---|---|---|---|
| 0 | GPIO-0 | - | Timing adapter APB 0 | SBCon I2C (touch screen) | FPGA - SCC registers |
| 1 | GPIO-1 | - | Timing adapter APB 1 | SBCon I2C (audio conf) | FPGA - I2S (audio) |
| 2 | GPIO-2 | - | Timing adapter APB 2 | FPGA PL022 (SPI2 for ADC) | FPGA – GPIO (System Ctrl + I/O) |
| 3 | GPIO-3 | - | - | FPGA PL022 (SPI Shield0) | UART0 |
| 4 | User AHB 0 | - | - | FPGA PL022 (SPI Shield1) | UART1 |
| 5 | User AHB 1 | - | - | FPGA SBCon (I2C - Shield0) | UART2 |
| 6 | User AHB 2 | - | - | FPGA SBCon (I2C – Shield1) | UART3 STUB |
| 7 | User AHB 3 | - | - | Reserved | UART4 STUB |
| 8 | USB and ethernet | - | - | FPGA – SBCon I2C (DDR4 EPROM) | UART5 |
| 9 | - | - | - | - | Reserved |
| 10 | - | - | - | - | CLCD config reg |
| 11 | - | - | - | - | RTC |
| 12 | - | - | - | - | - |
| 13 | - | - | - | - | - |
| 14 | - | - | - | - | - |
| 15 | - | - | - | - | - |
| PPC IRQ no. | 5 | 6 | 2 | 3 | 4 |

## 6.4 Corstone™ SSE-310 FVP memory components

The Corstone™ SSE-310 FVP includes the following memory components and their security access is controlled by the MPC.

**Table 6-8: Memory components**

| Name | Non-secure address range | Secure alias | Size |
|---|---|---|---|
| SRAM | `0x0100_0000 - 0x011F_FFFF` | `0x1100_0000 - 0x111F_FFFF` | 2MB |
| QSPI SRAM | `0x2800_0000 - 0x287F_FFFF` | `0x3800_0000 - 0x387F_FFFF` | 8MB |

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**Table 6-9: DDR regions**

| Name | Address range | IDAU region security | Size |
|---|---|---|---|
| DDR0 | `0x6000_0000 - 0x6FFF_FFFF` | NS | 256MB |
| DDR1 | `0x7000_0000 - 0x7FFF_FFFF` | S | 256MB |
| DDR2 | `0x8000_0000 - 0x8FFF_FFFF` | NS | 256MB |
| DDR3 | `0x9000_0000 - 0x9FFF_FFFF` | S | 256MB |
| DDR4 | `0xA000_0000 - 0xAFFF_FFFF` | NS | 256MB |
| DDR5 | `0xB000_0000 - 0xBFFF_FFFF` | S | 256MB |
| DDR6 | `0xC000_0000 - 0xCFFF_FFFF` | NS | 256MB |
| DDR7 | `0xD000_0000 - 0xDFFF_FFFF` | S | 256MB |

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

# 7. Arm® Corstone™-1000 FVP

This is an example FVP system as specified by the Corstone™-1000 architecture and system specifications.

Refer to the Arm Corstone-1000 Technical Overview for more information.

---

Note

You must have installed the MMEncrypt add-on package to use this FVP. It can be downloaded from Product Download Hub.

---

## 7.1 Corstone™-1000 FVP modeled components

The following components are present in the Corstone™-1000 FVP.

- Host controller:
  - Cortex-A35 64-bit 4x Cluster CPU
  - GIC400
  - Firewall
- MPS3 board:
  - SMSC 91C111 ethernet controller
  - PrimeCell RTC (PL031) module
  - Audio I2S and SBCon I2C as dummy stub modules
  - Two SD cards
  - DRAM 2GB
  - SRAM 32MB
  - PL050 KMI
  - PS2 keyboard
  - SP805 watchdog
- Secure enclave:
  - M0+ CPU
  - CryptoCell312
  - SE Flash (64KB up to 8MB)
  - NVM OTP memory for CC312
  - Firewall

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Note

The FVP does not model Arm® CoreSight™ components.

## 7.2 Run the Corstone™-1000 FVP with the software package

This FVP is intended to be used with the Arm reference design software package.

### About this task

The User Guide for the software package can be found at ARM Corstone1000.

### Procedure

1. To see a list of available configuration parameters, run the FVP with the `--list-params` option. Many of these parameters help to bring up bare-metal software on the FVP and are not required if the model is run with the Arm® firmware supplied.
2. This example command line launches the model with the Arm software package:

```
./FVP_Corstone-1000 \
    -C diagnostics=4 \
    -C se.trustedBootROMloader.fname="bl1.bin" \
    -C se.trustedSRAM_config=6 \
    -C se.BootROM_config="3" \
    -C se.nvm.raw_image="cc312_otp.bin" \
    --data board.flash0=corstone1000-aarch64-image-mps3.wic.nopt@0x68050000 \
    -C board.xnvm_size=64 \
    -C board.smsc_91c111.enabled=1 \
    -C board.hostbridge.userNetworking=true \
    -C board.se_flash_size=8192 \
    -C board.msd_mmc.p_mmc_file=mmc1.dat \
    -C board.msd_mmc.card_type="SD" \
    -C board.msd_mmc.p_fast_access=0 \
    -C board.msd_mmc_2.p_mmc_file=mmc2.dat \
    -C board.msd_mmc_2.card_type="SD" \
    -C board.msd_mmc_2.p_fast_access=0 \
    -C board.msd_mmc.p_max_block_count="0xFFFF" \
    -C board.msd_mmc_2.p_max_block_count="0xFFFF" \
    -C board.flashloader0.fname="flash0" \
    -C board.flashloader0.fnameWrite="flash0"
```

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

## 7.3 Corstone™-1000 board peripherals memory and interrupt map

The subsystem model is complemented by a range of peripherals. The board peripherals are representative of peripherals present on the board onto which the SoC is mounted.

**Table 7-1: Corstone™-1000 board peripherals memory and interrupt map**

| Component | Base address | Size | IRQ |
|---|---|---|---|
| Ethernet | `0x00_4010_0000` | 1M | 116 |
| SCC registers | `0x00_4000_0000` | 4KB | - |
| IO registers | `0x00_4001_0000` | 4KB | - |
| SD card 0 | `0x00_4030_0000` | 64KB | 149 |
| SD card 1 | `0x00_5000_0000` | 64KB | 147 |
| SE flash | `0x00_6001_0000` | 8MB | - |

## 7.4 Networking on Corstone™-1000 FVP

Steps to bring up the network on the FVP.

### Procedure

1. Set up the TAP interface on the host machine. For information, see TAP/TUN networking in the *Fast Models Reference Guide*.
2. Append the following parameters to the model:
   - `-C board.smsc_91c111.enabled=true`
   - `-C board.hostbridge.interfaceName="<TAP Interface Name>"`

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

# 8. Juno FVP3 model

This model implements the Juno Arm®v8 hardware platform, revision r0.

The Juno SoC is described in the Juno Arm Development Platform TRM.

The version number of the model, including the build number, can be obtained by running the model with the `--version` parameter.

For information about the changes in this release, see the Fast Models Portfolio release notes.

To provide feedback on the product, create a ticket at https://support.developer.arm.com.

## 8.1 Unimplemented features and model limitations

Some peripherals are unimplemented in the model, and others are implemented with limitations.

The model does not implement the following peripherals that are described in the Juno SoC ADP TRM:

- CoreSight™ and device discovery ROMs.
- CoreSight™ devices. The memory area is mapped as a dummy APB region.
- SoC Master signals.
- DFI phy config. It is mapped as a dummy APB region.
- PVT sensors are mapped as DummyAPB peripherals.
- PCIe and the associated SMMU, and the associated MSI mechanism.
- USB and the associated SMMU.
- I2C controllers and their associated peripherals.

The following peripherals are implemented but with limitations:

- The Mali™-T624 GPU is present in a form that permits a driver to load and make forward progress, but it does not render any output to the target surface or buffer.
- 2 x HDLCD controllers are present but the downstream physical ports and logic that would normally be part of selecting a suitable display format are not modeled. An additional Linux driver is available instead.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

## 8.2 Running the Juno FVP3 model

The model can be run through the Arm DS debug IDE or standalone.

Note

When running the model on a Linux host machine, ensure that it has at least 16GB of RAM, in addition to the normal Fast Models requirements. The Juno platform and model incorporate 8GB of DRAM.

### 8.2.1 Run the Juno FVP3 model in Arm DS

Arm DS can auto-detect and connect to the Juno FVP and offers access to the big.LITTLE™ cores individually or collectively. It is also possible to connect to the Cortex®-M3 core in the SCP.

Start the model with the `--iris-server` option, or the older `--cadi-server` option, then use **New > Model connection** to begin the process of auto-detection.

### 8.2.2 Run the Juno FVP3 model standalone

The Juno FVP can boot standard software stacks, for example Linux, with minimal modification.

Example command line:

```
$BUILD/Platforms/LISA/CSS/Build_Juno_FVP3/Linux64-Release-GCC-7.3/isim_system \
    --plugin $BUILD/PVLIB_HOME/plugins/Linux64_GCC-7.3/Crypto.so           \
    -C css.aon.scp.ROMloader.fname=juno-scp-rom.bin                        \
    -C board.flashloader0.fname=juno-nor-image.bin                         \
    -C board.base_clk_frequency=0x17D7840                                  \
    -C soc.pl011_uart0.unbuffered_output=1                                 \
    -C soc.pl011_uart1.unbuffered_output=1                                 \
    -C board.pl011_uart1.unbuffered_output=1                               \
    -C soc.scc.gpr0=0x03000000                                             \
    -C soc.scc.apps_alt_boot=0xBEC0000                                     \
    -C soc.scc.scp_alt_boot=0xABE40000                                     \
    -C board.mmc.p_mmc_file=optional-mmc-card.img                          \
    -C board.virtioblockdevice.image_path=optional-virtio-disk.img         \
    -C board.smsc_91c111.enabled=1                                         \
    -C board.hostbridge.userNetworking=<0|1>                               \
    -C board.hostbridge.interfaceName=<tap>
```

Note

- `juno-scp-rom.bin` is included with the model in this release.
- `juno-nor-image.bin` represents the contents of the NOR flash memory.

In hardware, there is a complex mechanism to present the NOR flash as a USB disk to allow replacement firmware, for example arm-trusted-firmware, boot loader, or Linux kernel, to be pushed to the board. Instead of modeling this, a script, `images/create_afs_image.py` is included to create the same memory image directly from the firmware components. Any optional MMC image should be less than 2GB.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

See User mode networking for more information about networking in Fast Models.

### 8.2.3 Run the create_afs_image.py script

This topic shows an example command line that boots Linux using U-Boot.

```
python3 create_afs_image.py juno-nor-image.bin                    \
${FW_SW_DIR}/fip-uboot.bin,-,0                                    \
${KERNEL_OUT_DIR}/arch/arm64/boot/Image,Image                     \
${KERNEL_OUT_DIR}/arch/arm64/boot/dts/arm/juno-fvp.dtb,board.dtb \
${FW_SW_DIR}/scp_bl1.bin,-,0x3e40000                              \
${FW_SW_DIR}/bl1.bin,-,0x3ec0000                                  \
uboot-script.img,BOOTSCR                                          \
./u-boot-saveenv.bin,-,0x3fc0000
```

The entry for each component of the NOR image consists of:

- The file on the host machine to add to the image.
- An optional alternative name to use for that file when the NOR metadata is generated. Use a '-' to not have an entry in the metadata at all, or leave blank to use the leaf name of the host file.
- An optional address at which to position the file in the NOR image. Those components shown with an address must use that address to place code and data at expected locations. Other components can be referenced by name so their address does not need to be fixed. Components with no fixed address are allocated sequentially after the previous component. There is no intelligent allocation of components to make best use of the available space. A warning occurs if the image exceeds the size of the NOR area.

In this command line:

- `FW_SW_DIR` points to the `SOFTWARE` directory of a standard Juno firmware bundle, for example those from http://releases.linaro.org/members/arm/platforms/20.01/ and `KERNEL_OUT_DIR` is the build output of a Linux kernel.
- `juno-fvp.dtb` is a modified version of the normal `juno.dts` that removes the unsupported hardware, for example PCIe, USB, and I2C audio, and adds some additional sections, for example the virtio disk and a helper mechanism for the HDLCD output.

  `juno-fvp.dts` is included alongside the model in this release and should be used to patch a standard Linux kernel to create the DTB.
- `BOOTSCR/uboot-script.img` allows you to make permanent changes to the boot sequence programmatically. One required change is typically to tell the kernel to boot from the virtio device `/dev/vda` instead of a regular disk at `/dev/sda`. With bigger kernels, it is also necessary to move the device tree up in memory, for example:

  ```
  echo "setenv fdt_addr 83000000" > uboot_script.txt
  echo "setenv bootargs \$bootargs root=/dev/vda" >> uboot_script.txt
  mkimage -T script -C none -n 'BOOTSCR' -d uboot-script.txt uboot_script.img
  ```

  If your disk image has no partition table, then `root=/dev/vda` is correct.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

If your disk image has a partition table then you need something like `root=/dev/vda2` to point at the correct partition to find the root fs. You can check this with:

```
fdisk -l optional-virtio-disk.img
```

- `u-boot-saveenv.bin` is an optional example of an alternative way to modify U-Boot settings by replacing the region of flash containing the saved U-Boot environment. This is not supplied, but can be created by using the U-Boot commands to modify the environment and save it to another part of memory to be dumped to disk.

## 8.3 Jump-start the application clusters on bare metal

It is possible to bring the application clusters out of reset without running firmware on the SCP. This is supported using parameters that allow the user to override the default configuration for the SCP control over Cortex®-A53 and Cortex®-A57 cluster reset.

Add the following lines as appropriate to the Juno configuration file:

```
-C css.aon.scp.scp_sc.a53_power_on=15
-C css.aon.scp.scp_sc.a57_power_on=3
```

The parameter is a bitfield describing which of the CPUs to turn on.

---

Note

If you are using a separate parameter file, the `-c` is not required. Specify each parameter on a new line in the file.

---

To supply an image at the same time, add the following:

```
-a css.cluster0.*=<your-image>
```

You can also target cluster1, and optionally any supported cpu in a cluster, for example `css.cluster0.cpu0`. Because the two clusters and the CPUs within a cluster share a memory interface, this rarely makes a difference.

## 8.4 Basic setup for the TZC-400 model on bare metal

The default state out of reset for the TZC-400 is to block all accesses to DRAM.

You can configure the TZC-400 through the programmer's view or by using command-line parameters. For many applications, you can ignore TrustZone® security for DRAM. In this case,

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

you must enable read/write access to DRAM from Secure and Non-Secure world in the application clusters. This can be achieved with the following parameters:

```
-C css.tzc400.rst_gate_keeper=0x0f
-C css.tzc400.rst_region_attributes_0=0xc000000f
-C css.tzc400.rst_region_id_access_0=0xffffffff
```

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

# 9. Arm® Neoverse™ N1 edge and Neoverse™ E1 edge reference design FVPs

This chapter describes the Arm® Neoverse™ N1 edge and Arm® Neoverse™ E1 edge reference design FVPs. These FVPs are collectively referred to as RD-N1-E1-edge FVPs.

A reference design is a collection of resources, including documentation, a software stack, and FVPs, that describe and model the design choices and performance for recommended configurations of a typical Arm®-based subsystem.

The RD-N1-E1-edge FVPs drive system architecture and software standardization. They provide software and binaries of proprietary firmware that reduce the amount of work that is required for SoC development.

---

Note

Arm is working to make more content available for these FVPs. To find out more about reference designs, or to contact Arm about them, see Neoverse Reference Design.

---

## 9.1 About the RD-N1-E1 FVPs

The RD-N1-E1-edge FVPs model many of the Arm® IP components in the RD-N1-E1-edge design.

The RD-N1-E1-edge FVPs model the following RD-N1-E1-edge configurations:

**Config1**
: N1 2xMP4, 512KB L2 cache per core, 4x2 mesh, 2xDMC.

**Config2**
: E1 2xMP8, 256KB L2 cache per core, 4x2 mesh, 2xDMC.

**Config3**
: Dual-chip. Two Config1 subsystems linked by CMN-600 CML.

The diagrams in 9.2 Block diagrams for RD-N1-E1-edge on page 73 show the IP components that RD-N1-E1-edge describes. Not all of these components are modeled by these FVPs. The following components are modeled:

- N1 2xMP4 or E1 2xMP8 cores.
- System Control Processor (SCP).
- Management Control Processor (MCP).
- CMN-600.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

- Multiple NIC-450 interconnects, although NIC-450 is replaced with a simple bus model.
- Memory access path towards DRAM, including DMC-620 memory controllers.

The following components are not modeled:

- CoreSight™ SoC.
- ELA-500.
- USB.

## 9.2 Block diagrams for RD-N1-E1-edge

The following diagrams show the composition of RD-N1-E1-edge systems.

**Figure 9-1: Block diagram for Neoverse N1 edge reference design**

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**Figure 9-2: Block diagram for Neoverse E1 edge reference design**

## 9.3 RD-N1-E1 FVP peripherals

The RD-N1-E1-edge FVPs include peripherals that the software payload requires to run.

These peripherals are organized in two layers:

**SoC**

The SoC peripherals represent peripherals that may be added to a compute subsystem in a SoC design.

**Board**

The board peripherals represent peripherals that may be present on the board onto which the SoC is mounted.

The RD-N1-E1-edge SoC model and board model are based on the Juno Arm® Development Platform (ADP).

In Config3, each compute subsystem has its own SoC and board layers. In this two-chip configuration, each chip is assigned the following memory regions:

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**Chip 0**

`0x000_0000_0000-0x3FF_FFFF_FFFF`

**Chip 1**

`0x400_0000_0000-0x7FF_FFFF_FFFF`

A sideband communication channel is required to coordinate multi-chiplet software boot over CMN-600. The FVP implements this using the MHU device, but Arm recommends using a solution such as I2C in hardware.

### 9.3.1 Memory map for RD-N1-E1-edge FVP SoC peripherals

This table shows the memory map for the SoC peripherals in the RD-N1-E1-edge FVPs.

Note

The SoC peripherals area in the RD_N1_E1_edge memory map is mapped to an Expansion AXI region. Therefore, these mappings are not defined in the reference design.

**Table 9-1: SoC peripherals memory map**

| Name | Base address | Size | Description |
|---|---|---|---|
| SMC interface | `0x00_0800_0000` | 368MB | Routed to board |
| SMMUv3 | `0x00_2B40_0000` | 1MB | - |
| PCIe config | `0x00_6000_0000` | 16MB | - |
| PCIe memory | `0x00_7000_0000` | 132MB | - |
| DMA MMU-400 | `0x00_7FB0_0000` | 64KB | - |
| HDLCD1 MMU-400 | `0x00_7FB1_0000` | 64KB | - |
| HDLCD0 MMU-400 | `0x00_7FB2_0000` | 64KB | - |
| DDR3 PHY 0 | `0x00_7FB6_0000` | 64KB | Dummy APB |
| DDR3 PHY 1 | `0x00_7FB7_0000` | 64KB | Dummy APB |
| DDR3 PHY 2 | `0x00_7FB8_0000` | 64KB | Dummy APB |
| DDR3 PHY 3 | `0x00_7FB9_0000` | 64KB | Dummy APB |
| SoC interconnect NIC-400 GPV | `0x00_7FD0_0000` | 1MB | - |
| Surge detector | `0x00_7FE5_0000` | 4KB | Dummy APB |
| TRNG | `0x00_7FE6_0000` | 4KB | - |
| Trusted non-volatile counters | `0x00_7FE7_0000` | 4KB | - |
| Trusted Root-Key storage | `0x00_7FE8_0000` | 4KB | - |
| Secure I2C | `0x00_7FE9_0000` | 256B | A Secure I2C component does not exist in the FVP. Instead, a PL061_GPIO is mapped as a dummy component. It is not functional as a GPIO. |
| DMA non-secure | `0x00_7FF0_0000` | 4KB | - |

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

| Name | Base address | Size | Description |
|---|---|---|---|
| DMA secure | 0x00_7FF1_0000 | 4KB | - |
| HDLCD1 | 0x00_7FF5_0000 | 4KB | - |
| HDLCD0 | 0x00_7FF6_0000 | 4KB | - |
| UART 1 | 0x00_7FF7_0000 | 4KB | - |
| UART 0 | 0x00_7FF8_0000 | 4KB | - |
| I2S | 0x00_7FF9_0000 | 1KB | An I2S component does not exist in the FVP. Instead, a PL061_GPIO is mapped as a dummy component. It is not functional as a GPIO. |
| I2C | 0x00_7FFA_0000 | 0x256B | An I2C component does not exist in the FVP. Instead, a PL061_GPIO is mapped as a dummy component. It is not functional as a GPIO. |
| PL352 | 0x00_7FFD_0000 | 4KB | PL354 |
| AP configuration | 0x00_7FFE_0000 | 4KB | GPR |
| System override registers | 0x00_7FFF_0000 | 4KB | - |
| PCIe memory | 0x05_0000_0000 | 12GB | - |

Note

The following devices are discoverable on the PCIe bus:

- Virtio block device x 2.
- AHCI controller with attached SATA disk.

## 9.3.2 Memory map for RD-N1-E1-edge FVP board peripherals

This table shows the memory map for the board peripherals in the RD-N1-E1-edge FVPs.

Note

The board peripherals area in the RD_N1_E1_edge memory map is mapped to an Expansion AXI region. Therefore, these mappings are not defined in the reference design.

**Table 9-2: Board peripherals memory map**

| Name | Base address | Size | Description |
|---|---|---|---|
| NOR Flash 0 | 0x00_0800_0000 | 64MB | - |
| NOR Flash 1 | 0x00_0C00_0000 | 64MB | - |
| NOR Flash 2 | 0x00_1000_0000 | 64MB | - |
| Ethernet | 0x00_1800_0000 | 64MB | SMSC 91C111 |
| System registers | 0x00_1C01_0000 | 64KB | - |
| MCI | 0x00_1C05_0000 | 64KB | PL180 |
| KMI 0 | 0x00_1C06_0000 | 64KB | PL050 |
| KMI 1 | 0x00_1C07_0000 | 64KB | PL050 |
| UART 0 | 0x00_1C09_0000 | 64KB | PL011 |

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

| Name | Base address | Size | Description |
|---|---|---|---|
| UART 1 | `0x00_1C0A_0000` | 64KB | PL011 |
| Watchdog | `0x00_1C0F_0000` | 64KB | SP805 |
| Dual timer | `0x00_1C11_0000` | 64KB | SP804 |
| Virtio block device | `0x00_1C13_0000` | 64KB | - |
| Virtio net device | `0x00_1C15_0000` | 64KB | - |
| RTC | `0x00_1C17_0000` | 64KB | PL031 |
| GPIO 0 | `0x00_1C1D_0000` | 64KB | PL061_GPIO is mapped as a dummy component in the FVP. It is not functional as a GPIO. |
| GPIO 1 | `0x00_1C1E_0000` | 64KB | PL061_GPIO is mapped as a dummy component in the FVP. It is not functional as a GPIO. |
| DRAM | `0x00_8000_0000` | 2GB | - |
| DRAM | `0x80_8000_0000` | 6GB | - |

### 9.3.3 Interrupt maps for RD-N1-E1-edge FVP

These tables show the interrupt IDs and sources for the FVP peripherals.

**Table 9-3: Interrupt map at the SoC layer**

| Interrupt ID | Source | Description |
|---|---|---|
| 147 | UART 0 | - |
| 148 | UART 1 | - |
| 171 | TRNG | - |

**Table 9-4: Interrupt map at the board layer**

| Interrupt ID | Source | Description |
|---|---|---|
| 111 | Ethernet | - |
| 132 | RTC | - |
| 133 | UART 0 | - |
| 134 | UART 1 | - |
| 140 | VFS2 | - |
| 202 | Virtio | - |
| 204 | Virtio net device | - |
| 228 | Watchdog | - |
| 229 | KMI 0 | - |
| 230 | Dual timer | Interrupts 0 and 1 |
| 231 | System registers ethernet IRQ | - |

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

# 10. Arm® Neoverse™ N2 reference design FVP

The Arm® Neoverse™ N2 reference design (RD-N2) Fixed Virtual Platform (FVP) models much of the Arm® IP in RD-N2. The FVP enables partners to develop ahead of hardware availability and to explore the design from a software perspective.

The FVP is used with the RD-N2 software package. For instructions on setting up and running the FVP, see the RD-N2 Software Bundle Readme.

## 10.1 About the RD-N2 FVP

The RD-N2 FVP models CFG32C4M, which is a single-chiplet system with the number of Arm® Neoverse™ N2 cores reduced to 16.

The FVP models the following IP components:

- Arm® Neoverse™ N2 MP1 cores.
- CMN-700.

  Note: The CMN-700 model supports RAS, which when enabled can impact system performance. To avoid this, the model disables RAS by default. To enable it, set the CMN700 model parameter `enable_ras` to true.

- Multiple NIC-450 interconnects.
- GIC-700.
- MMU-700.
- Arm® Cortex®-M7 SCP and MCP cores.

Note: The FVP does not model every component that RD-N2 describes. For example, the FVP does not model the CoreSight™ technology components.

The RD-N2 FVP drives system architecture and software standardization. The models provide software and binaries of proprietary firmware that reduce the amount of work that is required for SoC development.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

## 10.2 RD-N2 FVP peripherals

The RD-N2 FVP includes peripherals that the software payload requires to run.

The SoC peripherals area and board peripherals area in the RD-N2 memory map are mapped to an expansion AMBA® AXI region. Therefore, these mappings are not defined in the reference design.

The peripherals are organized into two layers:

**SoC**

The SoC peripherals represent peripherals that might be added to a compute subsystem in an SoC design. The RD-N2 SoC model is based on the Juno Arm® Development Platform (ADP).

**Board**

The board peripherals represent peripherals that might be present on the board onto which the SoC is mounted. The RD-N2 board model is based on the Juno ADP.

### 10.2.1 Memory map for RD-N2 FVP SoC peripherals

This table shows the memory map for the SoC peripherals in the RD-N2 FVP.

**Table 10-1: RD-N2 FVP SoC peripherals**

| Name | Base address | Size | Description |
|---|---|---|---|
| SMC0 interface | `0x00_0800_0000` | 64MB | Routed to the board. |
| SMC1 interface | `0x10_5000_0000` | 256MB | Routed to the board. |
| PCIe config | `0x10_1000_0000` | 256MB | - |
| PCIe memory | `0x00_6000_0000` | 512MB | - |
| DMA MMU-500 | `0x00_E00_0000` | 64KB | - |
| HDLCD1 MMU-500 | `0x00_E01_0000` | 64KB | - |
| HDLCD0 MMU-500 | `0x00_E02_0000` | 64KB | - |
| SoC interconnect NIC-400 GPV | `0x00_ED0_0000` | 1MB | - |
| Surge detector | `0x00_EE5_0000` | 4KB | Dummy APB. |
| TRNG | `0x00_EE6_0000` | 4KB | - |
| Trusted non-volatile counters | `0x00_EE7_0000` | 4KB | - |
| Trusted root-key storage | `0x00_EE8_0000` | 4KB | - |
| Secure I2C | `0x00_EE9_0000` | 512 bytes | There is no Secure I2C component in the FVP. Instead, a PL061 GPIO is mapped as a dummy component. |
| DDR PHY 0–31 | `0x00_EB0_0000` | 2MB | Dummy APB. |
| DMA secure | `0x00_EF0_0000` | 64KB | - |
| DMA non-secure | `0x00_EF1_0000` | 64KB | - |
| PCIE macro | `0x00_EF2_0000` | 64KB | - |
| PCIE root port | `0x00_EF3_0000` | 64KB | - |

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

| Name | Base address | Size | Description |
|---|---|---|---|
| HDLCD1 | `0x00_EF5_0000` | 4KB | - |
| HDLCD0 | `0x00_EF6_0000` | 4KB | - |
| UART 0 | `0x00_EF7_0000` | 4KB | - |
| UART 1 | `0x00_EF8_0000` | 4KB | - |
| I2S | `0x00_EF9_0000` | 1KB | There is no I2S component in the FVP. Instead, a PL061 GPIO is mapped as a dummy component. |
| I2C | `0x00_EFA_0000` | 256 bytes | There is no I2C component in the FVP. Instead, a PL061 GPIO is mapped as a dummy component. |
| PL354 | `0x00_EFD_0000` | 64KB | Arm® PrimeCell SMC dual SRAM memory interface (PL354). |
| System override registers | `0x00_EFF_0000` | 64KB | - |
| AP configuration | `0x00_EFE_0000` | 64KB | Granular Power Requester (GPR). |

### 10.2.2 Memory map for RD-N2 FVP board peripherals

This table shows the memory map for the board peripherals in the RD-N2 FVP.

**Table 10-2: RD-N2 FVP board peripherals**

| Name | Base address | Size | Description |
|---|---|---|---|
| NOR flash 0 | `0x00_0800_0000` | 64MB | - |
| NOR flash 1 | `0x10_5000_0000` | 64MB | - |
| NOR flash 2 | `0x10_5400_0000` | 64MB | - |
| Ethernet | `0x10_5C00_0000` | 64MB | Non-PCI Ethernet controller (SMSC 91C111) |
| System registers | `0x00_0C01_0000` | 64KB | - |
| SP810 Sysctrl | `0x00_0C02_0000` | 64KB | - |
| MCI | `0x00_0C05_0000` | 64KB | Arm® PrimeCell Multimedia Card Interface (PL180) |
| KMI 0 | `0x00_0C06_0000` | 64KB | Arm® PrimeCell PS2 Keyboard/Mouse Interface (PL050) |
| KMI 1 | `0x00_0C07_0000` | 64KB | Arm® PrimeCell PS2 Keyboard/Mouse Interface (PL050) |
| UART 0 | `0x00_0C09_0000` | 64KB | Arm® PrimeCell UART (PL011) |
| UART 1 | `0x00_0C0A_0000` | 64KB | Arm® PrimeCell UART (PL011) |
| Watchdog | `0x00_0C0F_0000` | 64KB | Arm® Watchdog Module (SP805) |
| Dual timer | `0x00_0C11_0000` | 64KB | Arm® Dual-Timer Module (SP804) |
| Virtio block device | `0x00_0C13_0000` | 64KB | - |
| Virtio net device | `0x00_0C15_0000` | 64KB | - |
| GPIO 2Wire (DVI) | `0x00_0C16_0000` | 64KB | Arm® PrimeCell General Purpose Input/Output (PL061) |
| RTC0 | `0x00_0C17_0000` | 64KB | Arm® PrimeCell Real Time Clock (PL031) |
| RTC1 | `0x00_0C18_0000` | 64KB | Arm® PrimeCell Real Time Clock (PL031) |
| GPIO 0 | `0x00_0C1D_0000` | 64KB | Arm® PrimeCell General Purpose Input/Output (PL061) |
| GPIO 1 | `0x00_0C1E_0000` | 64KB | Arm® PrimeCell General Purpose Input/Output (PL061) |
| DRAM | `0x00_8000_0000` | 2GB | - |

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

| Name | Base address | Size | Description |
|---|---|---|---|
| DRAM | 0x80_8000_0000 | 6GB | - |

### 10.2.3 Interrupt maps for RD-N2 FVP

These tables show the interrupt IDs and sources for the FVP peripherals.

**Table 10-3: Interrupt map at the SoC layer**

| Interrupt ID | Source |
|---|---|
| 403 | UART 0 |
| 404 | UART 1 |
| 427 | TRNG |

**Table 10-4: Interrupt map at the board layer**

| Interrupt ID | Source |
|---|---|
| 387 | RTC1 |
| 388 | RTC0 |
| 389 | UART0 (board) |
| 390 | UART1 (board) |
| 391 | KMI1 |
| 392 | GPIO0 |
| 393 | GPIO1 |
| 394 | I2C GPIO |
| 395 | MCIINTR0 |
| 396 | MCIINTR1 |
| 397 | SMSC 91C111 |
| 405 | HDLCD controller 0 |
| 406 | SMC PL354 interface 0 |
| 407 | SMC PL354 interface 1 |
| 413 | HDLCD controller 1 |
| 414 | SMMU combined Secure interrupt |
| 415 | SMMU combined Non-secure interrupt |
| 418–425 | DMA0 IRQ7–0 |
| 426 | DMA0 IRQ abort |
| 428–435 | DMA1 IRQ7–0 |
| 436 | DMA1 IRQ abort |
| 458 | Virtio block device |
| 460 | Virtio net |
| 483 | RTCC |
| 484 | WDT |

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

| Interrupt ID | Source |
|---|---|
| 485 | KMI0 |
| 486 | Dual timer |
| 487 | System registers |
| 488 | System register – USB |
| 489 | System register – Tile |
| 490 | System register – Push button |
| 491 | System register – Ethernet |

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

# 11. Arm® Neoverse™ V1 reference design FVP

To develop ahead of hardware availability and to explore the design from a software perspective, the Fixed Virtual Platform (FVP) models many of the Arm® IP in the RD-V1 design.

## 11.1 About the RD-V1 FVP

Two configurations of RD-V1 are supported.

- RD-V1 FVP models Config-M, a single-chiplet system with 16 Arm® Neoverse™ V1 cores.
- RD-V1 quad-chiplet FVP models a reduced-size variant of Config-XL, consisting of four compute subsystems linked by CMN-650 CML. It provides a functional model of a quad-chiplet system. Each subsystem contains four Arm® Neoverse™ V1 cores, for a total of 16 cores in the FVP (at full size, Config-XL has 4 x 32 cores).

The FVP models the following IP components:

- Arm® Neoverse™ V1 MP1.
- GIC-700.
- MMU-600.
- SCP.
- MCP.
- CMN-650.
- Multiple NIC-450 interconnects.
- Memory access path towards DRAM which includes a TrustZone® controller.

---

Note

The FVP does not model every component that RD-V1 describes. For example, it does not model the CoreSight™ technology components.

---

The RD-V1 FVP drives system architecture and software standardization. The models provide software and binaries of proprietary firmware that reduce the amount of work that is required for SoC development.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

## 11.2 RD-V1 FVP peripherals

The RD-V1 FVP includes peripherals that the software payload requires to run.

These peripherals are organized in two layers:

**SoC**

The SoC peripherals represent peripherals that can be added to a compute subsystem in a SoC design. The Arm® Neoverse™ V1 reference design SoC model is based on the Juno Arm Development Platform (ADP).

**Board**

The board peripherals represent peripherals that can be present on the board onto which the SoC is mounted. The RD-V1 board model is based on the Juno Arm Development Platform (ADP).

In the quad-chiplet system, each compute subsystem has SoC and Board layers dedicated to that subsystem. Addressing for each chip is defined in chapter 7.1, AP memory map, in the Arm® Neoverse™ V1 reference design Software Developer Guide.

**Chip 0** `0x000_0000_0000-0x3FF_FFFF_FFFF`
**Chip 1** `0x400_0000_0000-0x7FF_FFFF_FFFF`
**Chip 2** `0x800_0000_0000-0xBFF_FFFF_FFFF`
**Chip 3** `0xC00_0000_0000-0xFFF_FFFF_FFFF`

A sideband communication channel is required to coordinate multi-chiplet software boot over CMN-650. The FVP implements this using the MHU device, but Arm recommends using a solution such as $I^2C$ in hardware.

### Related information

Arm Neoverse v1 reference design Software Developer Guide

### 11.2.1 Memory map for RD-V1 FVP SoC peripherals

This table shows the memory map for the SoC peripherals in the RD-V1 FVP.

Note

The following devices are discoverable on the PCIe bus:

- Virtio block device x 2.
- AHCI controller with attached SATA disk.

**Table 11-1: SoC peripherals**

| Name | Base address | Size | Description |
|---|---|---|---|
| SMC interface | `0x00_0800_0000` | 368MB | Routed to Board |
| SMMUv3 | `0x00_2B40_0000` | 1MB | - |
| PCIe Config | `0x00_6000_0000` | 16MB | - |

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

| Name | Base address | Size | Description |
|---|---|---|---|
| PCIe Memory | 0x00_7000_0000 | 132MB | - |
| DMA MMU-400 | 0x00_7FB0_0000 | 64KB | - |
| HDLCD1 MMU-400 | 0x00_7FB1_0000 | 64KB | - |
| HDLCD0 MMU-400 | 0x00_7FB2_0000 | 64KB | - |
| DDR3 PHY 0 | 0x00_7FB6_0000 | 64KB | Dummy APB |
| DDR3 PHY 1 | 0x00_7FB7_0000 | 64KB | Dummy APB |
| DDR3 PHY 2 | 0x00_7FB8_0000 | 64KB | Dummy APB |
| DDR3 PHY 3 | 0x00_7FB9_0000 | 64KB | Dummy APB |
| SoC Interconnect NIC-400 GPV | 0x00_7FD0_0000 | 1MB | - |
| Surge detector | 0x00_7FE5_0000 | 4KB | Dummy APB |
| TRNG | 0x00_7FE6_0000 | 4KB | - |
| Trusted Non-volatile counters | 0x00_7FE7_0000 | 4KB | - |
| Trusted Root-Key storage | 0x00_7FE8_0000 | 4KB | - |
| Secure I2C | 0x00_7FE9_0000 | 256B | A Secure I2C component does not exist in the FVP. Instead, a PL061_GPIO is mapped. |
| DMA Non-secure | 0x00_7FF0_0000 | 4KB | - |
| DMA Secure | 0x00_7FF1_0000 | 4KB | - |
| HDLCD1 | 0x00_7FF5_0000 | 4KB | - |
| HDLCD0 | 0x00_7FF6_0000 | 4KB | - |
| UART 1 | 0x00_7FF7_0000 | 4KB | - |
| UART 0 | 0x00_7FF8_0000 | 4KB | - |
| I2S | 0x00_7FF9_0000 | 1KB | An I2S component does not exist in the FVP. Instead, a PL061_GPIO is mapped. |
| I2C | 0x00_7FFA_0000 | 256B | An I2C component does not exist in the FVP. Instead, a PL061_GPIO is mapped. |
| PL352 | 0x00_7FFD_0000 | 4KB | PL354 |
| AP configuration | 0x00_7FFE_0000 | 4KB | GPR |
| System override Registers | 0x00_7FFF_0000 | 4KB | - |
| PCIe Memory | 0x05_0000_0000 | 12GB | - |

### 11.2.2 Memory map for RD-V1 FVP board peripherals

This table shows the memory map for the board peripherals in the RD-V1 FVP.

Note

The SoC peripherals area and board peripherals area in the RD-V1 memory map are mapped to an Expansion AXI region. Therefore, these mappings are not defined in the reference design.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**Table 11-2: Board peripherals**

| Name | Base address | Size | Description |
|---|---|---|---|
| NOR Flash 0 | `0x00_0800_0000` | 64MB | - |
| NOR Flash 1 | `0x00_0C00_0000` | 64MB | - |
| NOR Flash 2 | `0x00_1000_0000` | 64MB | - |
| Ethernet | `0x00_1800_0000` | 64MB | SMSC 91C111 |
| System registers | `0x00_1C01_0000` | 64KB | - |
| MCI | `0x00_1C05_0000` | 64KB | PL180 |
| KMI 0 | `0x00_1C06_0000` | 64KB | PL050 |
| KMI 1 | `0x00_1C07_0000` | 64KB | PL050 |
| UART 0 | `0x00_1C09_0000` | 64KB | PL011 |
| UART 1 | `0x00_1C0A_0000` | 64KB | PL011 |
| Watchdog | `0x00_1C0F_0000` | 64KB | SP805 |
| Dual Timer | `0x00_1C11_0000` | 64KB | SP804 |
| Virtio Block Device | `0x00_1C13_0000` | 64KB | - |
| Virtio Net Device | `0x00_1C15_0000` | 64KB | - |
| RTC | `0x00_1C17_0000` | 64KB | PL031 |
| GPIO 0 | `0x00_1C1D_0000` | 64KB | PL061_GPIO |
| GPIO 1 | `0x00_1C1E_0000` | 64KB | PL061_GPIO |
| DRAM | `0x00_8000_0000` | 2GB | - |
| DRAM | `0x80_8000_0000` | 6GB | - |

### 11.2.3 Interrupt maps for RD-V1 FVP

These tables show the interrupt IDs and sources for the FVP peripherals.

**Table 11-3: Interrupt map at the SoC layer**

| Interrupt ID | Source | Description |
|---|---|---|
| 147 | UART 0 | - |
| 148 | UART 1 | - |
| 171 | TRNG | - |

**Table 11-4: Interrupt map at the board layer**

| Interrupt ID | Source | Description |
|---|---|---|
| 111 | Ethernet | - |
| 132 | RTC | - |
| 133 | UART 0 | - |
| 134 | UART 1 | - |
| 140 | VFS2 | - |
| 202 | Virtio | - |

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

| Interrupt ID | Source | Description |
|---|---|---|
| 204 | Virtio net device | - |
| 228 | Watchdog | - |
| 229 | KMI 0 | - |
| 230 | Dual Timer | Interrupts 0 and 1 |
| 231 | System registers Ethernet IRQ | - |

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

# 12. Configurable PCIe hierarchy

Configurable PCIe hierarchy allows you to create a customized PCIe hierarchy based on a JSON-formatted file.

To use it, add the following parameter to the command line before running the model, specifying the path to the PCIe hierarchy file:

```
-C pci.hierarchy_file_name=/path/to/hierarchy.json
```

Note

PCIe hierarchy files are supported by most Infrastructure Reference Design FVPs. To find out if your platform supports it, run it with the `--list-params` option to see if it has the `*.hierarchy_file_name` parameter.

## 12.1 JSON format for the hierarchy

To represent an endpoint device, a root port, or a switch, use the following JSON format.

An endpoint device is represented in the following format:

```
{
    "<device_type>/<device_name>": {
        "parameter1":<parameter_value>
        "parameter2":<parameter_value>
        "   ...    ":<       ...       >
    }
}
```

A root port is represented in the following format, where a __downstream__ parameter denotes a device connected downstream of the port:

```
{
    "rootport/<rootport_name>": {
        "parameter1":<parameter value>
        "parameter2":<parameter value>
        "   ...    ":<      ...      >
        "__downstream__":{
            "<device_type>/<device_name>":{
                "parameter3":<parameter value>
                "parameter4":<parameter value>
                "   ...    ":<      ...      >
            }
        }
    }
}
```

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

A switch is represented in the following format, where a __downstream__<device_number> parameter denotes the device connected downstream of the switch, with the downstream port at device number <device_number> on the switch's internal bus:

```
{
    "switch/<switch_name>": {
        "parameter1":<parameter value>
        "parameter2":<parameter value>
        "    ...    ":<      ...      >
        "__downstream__<device_number1>": {
            "<device_type>/<device_name>":{
                "parameter3":<parameter value>
                "parameter4":<parameter value>
                "    ...    ":<      ...      >
            }
        },
        "__downstream__<device_number2>"{
            "<device_type>/<device_name>":{
                "parameter5":<parameter value>
                "parameter6":<parameter value>
                "    ...    ":<      ...      >
            }
        }
    }
}
```

## 12.2 Common PCIe endpoint parameters

A PCIe endpoint is the common PCIe frontend for all endpoint-type devices in the hierarchy, namely AHCI controller, host bridge, and SMMU test engine, which are described later in this chapter.

The following table describes the parameters for PCIe-related features of endpoints:

**Table 12-1: Common endpoint parameters**

| Parameter | Description | Type | Range |
|---|---|---|---|
| `device` | Device number of the endpoint. | int | 0-31 |
| `function` | Function number for the endpoint. | int | 0-7 |
| `vendor_id` | PCI vendor ID. | int | `0-0xFFFE` |
| `device_id` | PCI device ID. | int | `0-0xFFFE` |
| `base_class` | PCI base class. | int | 0-255 |
| `sub_class` | PCI sub class. | int | 0-255 |
| `bar0_64bit` | If BAR 0 is 64 bits wide, if region size is nonzero. | bool | - |
| `bar1_64bit` | If BAR 1 is 64 bits wide, if region size is nonzero. | bool | - |
| `bar2_64bit` | If BAR 2 is 64 bits wide, if region size is nonzero. | bool | - |
| `bar3_64bit` | If BAR 3 is 64 bits wide, if region size is nonzero. | bool | - |
| `bar4_64bit` | If BAR 4 is 64 bits wide, if region size is nonzero. | bool | - |

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

| Parameter | Description | Type | Range |
|---|---|---|---|
| `bar0_log2_size` | Log2 of the size of the region pointed to by BAR 0. Zero is reserved which means bar is not used [7]. | int | 0-63 |
| `bar1_log2_size` | Log2 of the size of the region pointed to by BAR 1. Zero is reserved which means bar is not used. | int | 0-63 |
| `bar2_log2_size` | Log2 of the size of the region pointed to by BAR 2. Zero is reserved which means bar is not used. | int | 0-63 |
| `bar3_log2_size` | Log2 of the size of the region pointed to by BAR 3. Zero is reserved which means bar is not used. | int | 0-63 |
| `bar4_log2_size` | Log2 of the size of the region pointed to by BAR 4. Zero is reserved which means bar is not used. | int | 0-63 |
| `bar5_log2_size` | Log2 of the size of the region pointed to by BAR 5. Zero is reserved which means bar is not used. | int | 0-63 |
| `uses_interrupt` | Enable support for legacy interrupts. | bool | - |
| `interrupt_pin_index` | Interrupt pin used by this function. 1 is INTA, 2 is INTB, 3 is INTC, 4 is INTD. | int | 1-4 |
| `multi_function` | Set to true if this endpoint is part of a multi-function device. | bool | - |
| `pcie_version` | PCIe version, bits[3:0] in capabilities register. 1 is PCIe 3.0. 2 is PCIe 4.0. | int | 1-15 |
| `prog_iface` | PCI programming interface. | int | 0-15 |
| `rev_id` | PCI revision ID. | int | 0-15 |
| `subsys_id` | PCI subsystem ID. | int | `0-0xFFFE` |
| `subsys_vendor_id` | PCI subsystem vendor ID. | int | `0-0xFFFE` |
| `express_capability_device_type` | PCI Express Capabilities Device Type, bits[7:4] in capabilities register. 0 is PCIe EndPoint, 9 is RCiEP. | int | 0-15 |
| `msix_support` | Enable device support for MSI-X. | bool | - |
| `msix_pba_bar` | BAR used by MSI-X pending bit array. | int | 0-5 |
| `msix_table_bar` | BAR used by MSI-X table. | int | 0-5 |
| `msix_table_size` | Size of tables for MSI-X. | int | 0-2048 |
| `power_mgmt_capability` | Device supports power-management capabilities. | bool | - |
| `pasid_supported` | If set, then the PCIe device can emit PASID (SubstreamIDs). | bool | - |
| `ext_fmt_field_supported` | Enable extended format field support. | bool | - |
| `extended_tag_supported` | Extended tag field support. | bool | - |
| `link_port_number` | Port number for PCIe link. | int | 0-255 |
| `pri_supported` | If set, then the PCIe function supports Page Request Interface (requires ATS). | bool | - |
| `rber_supported` | Enable role-based error reporting. | bool | - |
| `slot_and_root_status_msi_idx` | MSI index for reporting slot and root status changes. | int | 0-2047 |
| `tag_10bit_completer_supported` | Enable 10-bit tag completer support. | bool | - |
| `tag_10bit_requester_supported` | Enable 10-bit tag requester support. | bool | - |
| `aspm_optionality_compliant` | Enable ASPM optionality compliance. | bool | - |

[7] Parameter value is fixed and cannot be overridden.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

| Parameter | Description | Type | Range |
|---|---|---|---|
| `ats_supported` | If set, then the PCIe function supports Address Translation Services (ATS). | bool | - |

## 12.3 AHCI controller parameters

The Advanced Host Controller Interface (AHCI), is a technical standard for an interface that enables software to communicate with Serial ATA (SATA) devices.

Default values for endpoint parameters:

- `device`: 0
- `function`: 0
- `vendor_id`: `0xABC`*
- `device_id`: `0xACED`*
- `base_class`: `0x1`* (device class for mass storage)
- `sub_class`: `0x6`* (sub class for SATA)
- `bar0_64bit`: false
- `bar1_64bit`: false
- `bar2_64bit`: false
- `bar3_64bit`: false
- `bar4_64bit`: false
- `bar0_log2_size`: 13
- `bar1_log2_size`: 13
- `bar2_log2_size`: 12
- `bar3_log2_size`: 13
- `bar4_log2_size`: 12
- `bar5_log2_size`: 13
- `uses_interrupt`: false*
- `interrupt_pin_index`: 1
- `multi_function`: false
- `pcie_version`: 2
- `prog_iface`: `0x1`*
- `rev_id`: `0x1`*
- `subsys_id`: `0x2`*
- `subsys_vendor_id`: `0x13B5`*

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

- `express_capability_device_type`: 0
- `msix_support`: true*
- `msix_pba_bar`: 4*
- `msix_table_bar`: 2*
- `msix_table_size`: 1*
- `power_mgmt_capability`: true
- `pasid_supported`: false
- `ext_fmt_field_supported`: true
- `extended_tag_supported`: true
- `link_port_number`: 0
- `pri_supported`: false
- `rber_supported`: true
- `slot_and_root_status_msi_idx`: 0
- `tag_10bit_completer_supported`: true
- `tag_10bit_requester_supported`: true
- `aspm_optionality_compliant`: true
- `ats_supported`: false

Note

* indicates parameter values that are fixed and cannot be overridden.

**Table 12-2: AHCI parameters and their default values**

| Parameter | Description | Default value |
|---|---|---|
| `force_mode` | Force disk to report support for at most PIO/DMA/NCQ mode (only for testing/bring-up purposes). PIO mode is always supported. Use NCQ for maximum performance. | "NCQ" |
| `image_path` | Comma-separated list of 0-32 disk images. Each image represents one SATA disk which is connected to one port of the AHCI controller. Empty list elements are allowed and result in a SATA port that has no disk attached. An empty string (default) means one SATA port with no disk attached. | "" |
| `run_async` | Do host I/O in a background thread asynchronously. Enabling this parameter makes the simulation non-deterministic and may or may not improve performance. | false |

## Example

```
"ahci/ahci0": {
        "device": 10,
        "function": 0,
        "image_path": "/path/to/image",
        "express_capability_device_type": 9
    }
```

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

## 12.4 Host bridge parameters

On real systems, the host bridge connects the tree of PCI buses, which are internally connected with PCI-to-PCI bridges, to the rest of the system. On FVPs, it is merely a placeholder representation, and does not provide any functionality.

Default values for endpoint parameters:

- `device`: 0
- `function`: 0
- `vendor_id`: `0x13B5`
- `device_id`: 0
- `base_class`: `0x6*`
- `sub_class`: 0
- `bar0_64bit`: false
- `bar1_64bit`: false
- `bar2_64bit`: false
- `bar3_64bit`: false
- `bar4_64bit`: false
- `bar0_log2_size`: 12
- `bar1_log2_size`: 0
- `bar2_log2_size`: 0
- `bar3_log2_size`: 0
- `bar4_log2_size`: 0
- `bar5_log2_size`: 0
- `uses_interrupt`: false
- `interrupt_pin_index`: 1
- `multi_function`: false
- `pcie_version`: 2
- `prog_iface`: `0xf`
- `rev_id`: `0xf`
- `subsys_id`: `0xf`
- `subsys_vendor_id`: `0x13B5`
- `express_capability_device_type`: 9
- `msix_support`: false
- `msix_pba_bar`: 6

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

- `msix_table_bar`: 6
- `msix_table_size`: 1
- `power_mgmt_capability`: true
- `pasid_supported`: false
- `ext_fmt_field_supported`: true
- `extended_tag_supported`: true
- `link_port_number`: 0
- `pri_supported`: true
- `rber_supported`: true
- `slot_and_root_status_msi_idx`: 0
- `tag_10bit_completer_supported`: true
- `tag_10bit_requester_supported`: true
- `aspm_optionality_compliant`: true

Note

* indicates parameter values that are fixed and cannot be overridden.

### Example

```
"hostbridge/hb": {
        "device": 0,
        "function": 0,
    }
```

## 12.5 SMMU test engine parameters

SMMUv3TestEngine is a PCIe device used to generate various stimuli that can help test SMMU integration in the system.

Default values for endpoint parameters:

- `device`: 0
- `function`: 0
- `vendor_id`: `0x13B5`*
- `device_id`: `0xFF80`
- `base_class`: `0xFF`*
- `sub_class`: 0
- `bar0_64bit`: true

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

- `bar1_64bit`: false
- `bar2_64bit`: true
- `bar3_64bit`: false
- `bar4_64bit`: true
- `bar0_log2_size`: 18
- `bar1_log2_size`: 0
- `bar2_log2_size`: 15*
- `bar3_log2_size`: 0
- `bar4_log2_size`: 12*
- `bar5_log2_size`: 0
- `uses_interrupt`: false
- `multi_function`: false
- `pcie_version`: 2
- `prog_iface`: `0x0`
- `rev_id`: `0xf`
- `subsys_id`: `0x0`
- `subsys_vendor_id`: `0x13B5`*
- `express_capability_device_type`: 9
- `msix_support`: true
- `msix_pba_bar`: 4
- `msix_table_bar`: 2
- `msix_table_size`: 2048
- `power_mgmt_capability`: true
- `pasid_supported`: false
- `ext_fmt_field_supported`: true
- `extended_tag_supported`: true
- `link_port_number`: 0
- `pri_supported`: true
- `rber_supported`: true
- `slot_and_root_status_msi_idx`: 0
- `tag_10bit_completer_supported`: true
- `tag_10bit_requester_supported`: true
- `aspm_optionality_compliant`: true

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Note

* indicates parameter values that are fixed and cannot be overridden.

### Example

```
"smmuv3testengine/smmute": {
        "device": 0,
        "function": 0,
    }
```

## 12.6 Root port parameters

The following root port parameters are available.

**Table 12-3: Root port parameters**

| Parameter | Description | Type | Range | Default value |
|---|---|---|---|---|
| device_number | Device number on this bus. | int | 0-31 | 0 |
| device_id | PCI device ID. | int | 0-0xFFFE | 0xdef |
| vendor_id | PCI vendor ID. | int | 0-0xFFFE | 0x13B5 |
| acs_supported | Access control services supported. | bool | - | false |
| aspm_optionality_compliant | Enable ASPM optionality compliance. | bool | - | true |
| ext_fmt_field_supported | Enable extended format field support. | bool | - | true |
| extended_tag_supported | Extended tag field support. | bool | - | true |
| link_port_number | Port number for PCIe link. | int | 0-255 | 0 |
| pcie_version | PCIe version, bits[3:0] in capabilities register. 1 is PCIe 3.0. 2 is PCIe 4.0. | int | 1-15 | 2 |
| rber_supported | Enable role-based error reporting. | bool | - | true |
| slot_and_root_status_msi_idx | MSI index for reporting slot and root status changes. | int | 0-2047 | 0 |
| tag_10bit_completer_supported | Enable 10-bit tag completer support. | bool | - | true |
| tag_10bit_requester_supported | Enable 10-bit tag requester support. | bool | - | true |

### Example

```
"rootport/rootport0": {
    "device_number": 5,
    "__downstream__": {
        "ahci/ahci1": {
        "device": 0,
        "function": 0,
        "image_path": "/path/to/image"
        }
    }
```

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

```
}
```

## 12.7 Switch parameters

This topic lists the switch parameters with their default values and gives an example.

**Table 12-4: Switch parameters**

| Parameter | Description | Type | Range | Default value |
|---|---|---|---|---|
| `device_number` | Device number on this bus. | int | 0-31 | 0 |
| `acs_supported` | Access control services supported. | bool | - | false |

### Example

```
"switch/switch0": {
            "device_number": 0,
            "__downstream__10": {
                "ahci/ahci4": {...}
            },
            "__downstream__20": {
                "ahci/ahci2": {...}
            }
        }
```

## 12.8 Root bridge parameters

The term *root bridge* is taken from the PCI Firmware Specification Revision 3.x. It refers to an abstraction for the platform's PCI config and memory I/O access mechanism. A root bridge can be used to create multiple ECAM and MMIO regions within the system's ECAM and MMIO limits.

**Table 12-5: Root bridge parameters**

| Parameter | Description | Type | Range |
|---|---|---|---|
| `ecam_start` | ECAM base address. | int | Min: <System ECAM base><br><br>Max:<System ECAM end - 1MB> |
| `ecam_end_incl` | ECAM end address. | int | Min:<System ECAM base + 1MB><br><br>Max:<System ECAM end> |
| `mem32_start` | 32-bit memory window base address. | int | Min:<System MEM32 base><br><br>Max:<System MEM32 end> |
| `mem32_end_incl` | 32-bit memory window end address. | int | Min:<System MEM32 base><br><br>Max:<System MEM32 end> |
| `mem64_start` | 64-bit memory window base address. | int | Min:<System MEM64 base><br><br>Max:<System MEM64 end> |

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

| Parameter | Description | Type | Range |
|---|---|---|---|
| `mem64_end_incl` | 64-bit memory window base address. | int | Min:<System MEM64 base><br><br>Max:<System MEM64 end> |
| `ecam_start_bus_number` | ECAM start bus number. | int | Min:<System start bus><br><br>Max:<System end bus><br><br>**Note:**<br>Each bus consumes 1MB of memory, so each ECAM should have enough space (`ecam_start` - `ecam_end_incl`) to accommodate the total number of buses (`start_bus` - `end_bus`) allocated to it. |

### Example

```
"rootbridge/rb0": {
        "ecam_start": 0x60000000,
        "ecam_end_incl": 0x67ffffff,
        "mem32_start": 0x70000000,
        "mem32_end_incl": 0x73ffffff,
        "mem64_start": 0x500000000,
        "mem64_end_incl": 0x67fffffff,
        "ecam_start_bus_number": 0x0,

        "__downstream__": {
            "ahci/ahci0": {
                "device": 30,
                "function": 0,
                "image_path": "",
                "express_capability_device_type": 9
            },
            "ahci/ahci1": {
                "device": 31,
                "function": 0,
                "express_capability_device_type": 9,
                "image_path": ""
            }
        }
    }
```

## 12.9 Exerciser

Exerciser is a PCIe device used to generate various stimuli that can help test PCIe/SMMU integration in the system.

**Table 12-6: Default values for endpoint parameters**

| Endpoint parameter | Default value |
|---|---|
| `device` | 0 |
| `function` | 0 |
| `vendor_id` | `0x13B5` |
| `device_id` | `0xED01` |
| `base_class` | `0xED` |

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

| Endpoint parameter | Default value |
|---|---|
| sub_class | 0 |
| bar0_64bit | false |
| bar1_64bit | false |
| bar2_64bit | false |
| bar3_64bit | false |
| bar4_64bit | false |
| bar0_log2_size | 12 |
| bar1_log2_size | 14 |
| bar2_log2_size | 15 |
| bar3_log2_size | 0 |
| bar4_log2_size | 0 |
| bar5_log2_size | 12 |
| uses_interrupt | true |
| interrupt_pin_index | 1 |
| multi_function | false |
| pcie_version | 2 |
| prog_iface | 0 |
| rev_id | 0 |
| subsys_id | 0 |
| subsys_vendor_id | 0x13B5 |
| express_capability_device_type | 0 |
| msix_support | true |
| msix_pba_bar | 4 |
| msix_table_bar | 2 |
| msix_table_size | 2048 |
| power_mgmt_capability | true |
| pasid_supported | false |
| ext_fmt_field_supported | true |
| extended_tag_supported | true |
| link_port_number | 0 |
| pri_supported | true |
| rber_supported | true |
| slot_and_root_status_msi_idx | 0 |
| tag_10bit_completer_supported | true |
| tag_10bit_requester_supported | true |
| aspm_optionality_compliant | true |
| ats_supported | true |
| error_injection_supported | false |
| aer_supported | false |

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

### Exerciser parameter and default value

**memory_bar**

BAR index to be used for exerciser memory access. Type: int, min: 0, max: 5, default: 1.

### Example

```
"exerciser/exerciser0": {
"device": 1,
"function": 0,
"interrupt_pin_index": 1,
"memory_bar": 3,
"express_capability_device_type": 0
}
```

## 12.10 Example hierarchy for a single ECAM configuration

The absence of a root bridge before the beginning of the device hierarchy implies a single ECAM configuration.

```
{
    "ahci/ahci0": {
        "device": 30,
        "function": 0,
        "image_path": "",
        "express_capability_device_type": 9
    },
    "smmuv3testengine/smmuv3testengine3": {
        "device": 15,
        "function": 0,
        "express_capability_device_type": 9
    },
    "rootport/rootport0": {
        "device_number": 2,
        "__downstream__": {
            "ahci/ahci1": {
            "device": 0,
            "function": 0,
            "image_path": "",
            "multi_function": true,
            },
        "ahci/ahci2": {
            "device": 0,
            "function": 1,
            "image_path": ""
            }
        }
    },
    "rootport/rootport1": {
        "device_number": 5,
        "__downstream__": {
            "ahci/ahci3": {
            "device": 0,
            "function": 0,
            "image_path": ""
            }
        }
    },
    "rootport/rootport2": {
        "device_number": 6,
        "__downstream__": {
            "switch/switch0": {
```

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

```
                "device_number": 0,
                "__downstream__10": {
                    "ahci/ahci4": {}
                },
                "__downstream__20": {
                    "ahci/ahci5": {}
                }
            }
        }
    }
}
```

## 12.11 Example hierarchy for two ECAM configuration

This example shows the JSON file for two ECAMs.

```
{
    "rootbridge/rb0": {
        "ecam_start": 0x60000000,
        "ecam_end_incl": 0x67ffffff,
        "mem32_start": 0x70000000,
        "mem32_end_incl": 0x73ffffff,
        "mem64_start": 0x500000000,
        "mem64_end_incl": 0x67fffffff,
        "ecam_start_bus_number": 0x0,

        "__downstream__": {
            "ahci/ahci0": {
                "device": 30,
                "function": 0,
                "image_path": "",
                "express_capability_device_type": 9
            },
            "ahci/ahci1": {
                "device": 31,
                "function": 0,
                "express_capability_device_type": 9,
                "image_path": ""
            }
        }
    },
    "rootbridge/rb1": {
        "ecam_start": 0x68000000,
        "ecam_end_incl": 0x6fffffff,
        "mem32_start": 0x74000000,
        "mem32_end_incl": 0x77ffffff,
        "mem64_start": 0x680000000,
        "mem64_end_incl": 0x7ffffffff,
        "ecam_start_bus_number": 0x1,

        "__downstream__": {
            "ahci/ahci0": {
                "device": 2,
                "function": 0,
                "image_path": "/path/to/image",
                "express_capability_device_type": 9
            }
        }
    }
}
```

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

# 13. Base Platform FVPs

This chapter describes the Base Platform FVPs contained in the FVP Standard Library and the Arm Architecture Envelope Models (AEM) FVPs.

For the Base Platform memory map, see Base Platform memory map in the *Fast Models Reference Guide*.

## 13.1 FVP_Base_AEMv8R

List of instances in FVP_Base_AEMv8R.

### FVP_Base_AEMv8R instances

**`FVP_Base_AEMv8R`**

Base Platform Compute Subsystem for AEMv8RMPCT.
Type: `FVP_Base_AEMv8R`.

**`FVP_Base_AEMv8R.bp`**

Peripherals and address map for the Base Platform.
Type: `BasePlatformPeripherals`.

**`FVP_Base_AEMv8R.bp.Timer_0_1`**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**`FVP_Base_AEMv8R.bp.Timer_0_1.clk_div0`**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**`FVP_Base_AEMv8R.bp.Timer_0_1.clk_div1`**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**`FVP_Base_AEMv8R.bp.Timer_0_1.counter0`**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**`FVP_Base_AEMv8R.bp.Timer_0_1.counter1`**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_AEMv8R.bp.Timer_2_3**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_AEMv8R.bp.Timer_2_3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_AEMv8R.bp.Timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_AEMv8R.bp.Timer_2_3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_AEMv8R.bp.Timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_AEMv8R.bp.ap_refclk**

ARM Generic Timer.
Type: `MemoryMappedGenericTimer`.

**FVP_Base_AEMv8R.bp.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**FVP_Base_AEMv8R.bp.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_AEMv8R.bp.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_AEMv8R.bp.clock300MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_AEMv8R.bp.clock32KHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_AEMv8R.bp.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_AEMv8R.bp.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_AEMv8R.bp.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_AEMv8R.bp.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_AEMv8R.bp.dmc**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_AEMv8R.bp.dmc_phy**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_AEMv8R.bp.dummy_local_dap_rom**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_AEMv8R.bp.dummy_ram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_AEMv8R.bp.dummy_usb**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_AEMv8R.bp.exclusive_monitor**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_Base_AEMv8R.bp.flash0**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_AEMv8R.bp.flash1**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_AEMv8R.bp.flashloader0**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_AEMv8R.bp.flashloader1**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_AEMv8R.bp.generic_watchdog**

ARM Generic Watchdog.
Type: `MemoryMappedGenericWatchdog`.

**FVP_Base_AEMv8R.bp.hdlcd0**

ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).
Type: `PL370_HDLCD`.

**FVP_Base_AEMv8R.bp.hdlcd0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_AEMv8R.bp.hdlcd0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_AEMv8R.bp.hdlcd0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_AEMv8R.bp.hdlcd0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_AEMv8R.bp.hdlcd0_labeller**

Type: `Labeller`.

**FVP_Base_AEMv8R.bp.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_Base_AEMv8R.bp.mmc**

Generic Multimedia Card.
Type: `MMC`.

**FVP_Base_AEMv8R.bp.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_AEMv8R.bp.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_AEMv8R.bp.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_AEMv8R.bp.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_AEMv8R.bp.mpe**

Encrypt memory transactions for each encryption context with an independent key to prevent mismatch access.
Type: `PVMemoryProtectionEngine`.

**FVP_Base_AEMv8R.bp.nontrustedrom**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_AEMv8R.bp.nontrustedromloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_AEMv8R.bp.ns_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_AEMv8R.bp.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_AEMv8R.bp.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_AEMv8R.bp.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_AEMv8R.bp.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_AEMv8R.bp.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_AEMv8R.bp.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_AEMv8R.bp.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_AEMv8R.bp.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_AEMv8R.bp.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_AEMv8R.bp.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_AEMv8R.bp.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_AEMv8R.bp.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_AEMv8R.bp.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_AEMv8R.bp.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_AEMv8R.bp.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_AEMv8R.bp.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_AEMv8R.bp.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_AEMv8R.bp.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_AEMv8R.bp.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_AEMv8R.bp.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_AEMv8R.bp.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ClockTimerThread`.

**FVP_Base_AEMv8R.bp.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_AEMv8R.bp.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_AEMv8R.bp.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_AEMv8R.bp.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

**FVP_Base_AEMv8R.bp.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_AEMv8R.bp.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_AEMv8R.bp.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_AEMv8R.bp.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_AEMv8R.bp.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_Base_AEMv8R.bp.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_AEMv8R.bp.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_AEMv8R.bp.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_AEMv8R.bp.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_AEMv8R.bp.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_AEMv8R.bp.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_AEMv8R.bp.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_AEMv8R.bp.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_AEMv8R.bp.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_Base_AEMv8R.bp.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

**FVP_Base_AEMv8R.bp.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_AEMv8R.bp.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_AEMv8R.bp.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_AEMv8R.bp.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_AEMv8R.bp.pl111_clcd_labeller**

Type: `Labeller`.

**FVP_Base_AEMv8R.bp.pl180_mci**

ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

**FVP_Base_AEMv8R.bp.ps2keyboard**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_AEMv8R.bp.ps2keyboard.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_AEMv8R.bp.ps2keyboard.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_AEMv8R.bp.ps2keyboard.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_AEMv8R.bp.ps2keyboard.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_AEMv8R.bp.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

**FVP_Base_AEMv8R.bp.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_AEMv8R.bp.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_AEMv8R.bp.ps2mouse.ps2_clocktimer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_AEMv8R.bp.ps2mouse.ps2_clocktimer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_AEMv8R.bp.psram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_AEMv8R.bp.refcounter**
Memory Mapped Counter Module for Generic Timers.
Type: `MemoryMappedCounterModule`.

**FVP_Base_AEMv8R.bp.reset_or**
Or Gate.
Type: `OrGate`.

**FVP_Base_AEMv8R.bp.rl_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_AEMv8R.bp.rt_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_AEMv8R.bp.s_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_AEMv8R.bp.secureDRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_AEMv8R.bp.secureSRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_AEMv8R.bp.secureflash**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_AEMv8R.bp.secureflashloader**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_AEMv8R.bp.smsc_91c111**

SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_Base_AEMv8R.bp.sp805_wdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_AEMv8R.bp.sp805_wdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_AEMv8R.bp.sp805_wdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_AEMv8R.bp.sp805_wdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_AEMv8R.bp.sp805_wdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_AEMv8R.bp.sp810_sysctrl**

Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_Base_AEMv8R.bp.sp810_sysctrl.clkdiv_clk0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_AEMv8R.bp.sp810_sysctrl.clkdiv_clk1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_AEMv8R.bp.sp810_sysctrl.clkdiv_clk2**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_AEMv8R.bp.sp810_sysctrl.clkdiv_clk3**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_AEMv8R.bp.sram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_AEMv8R.bp.terminal_0**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_AEMv8R.bp.terminal_1**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_AEMv8R.bp.terminal_2**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_AEMv8R.bp.terminal_3**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_AEMv8R.bp.trusted_key_storage**

Trusted Root-Key Storage unit.
Type: `RootKeyStorage`.

**FVP_Base_AEMv8R.bp.trusted_nv_counter**

Trusted Non-Volatile Counter unit.
Type: `NonVolatileCounter`.

**FVP_Base_AEMv8R.bp.trusted_rng**

Random Number Generator unit.
Type: `RandomNumberGenerator`.

**FVP_Base_AEMv8R.bp.trusted_watchdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_AEMv8R.bp.trusted_watchdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_AEMv8R.bp.trusted_watchdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_AEMv8R.bp.trusted_watchdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_AEMv8R.bp.trusted_watchdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_AEMv8R.bp.tzc_400**

TrustZone Address Space Controller.
Type: `TZC_400`.

**FVP_Base_AEMv8R.bp.ve_sysregs**

Type: `VE_SysRegs`.

**FVP_Base_AEMv8R.bp.vedcc**

Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

**FVP_Base_AEMv8R.bp.virtio_net**

VirtioNet device over MMIO transport.
Type: `VirtioNetMMIO`.

**FVP_Base_AEMv8R.bp.virtio_net_labeller**

Type: `Labeller`.

**FVP_Base_AEMv8R.bp.virtio_rng**

VirtioEntropy device over MMIO transport.
Type: `VirtioEntropyMMIO`.

**FVP_Base_AEMv8R.bp.virtioblockdevice**

VirtioBlock device over MMIO transport.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `VirtioBlockMMIO`.

**FVP_Base_AEMv8R.bp.virtioblockdevice_labeller**

Type: `Labeller`.

**FVP_Base_AEMv8R.bp.virtiop9device**

virtio P9 server.
Type: `VirtioP9Device`.

**FVP_Base_AEMv8R.bp.virtiop9device_labeller**

Type: `Labeller`.

**FVP_Base_AEMv8R.bp.vis**

Display window for VE using Visualisation library.
Type: `VEVisualisation`.

**FVP_Base_AEMv8R.bp.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_Base_AEMv8R.bp.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_AEMv8R.bp.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_AEMv8R.bp.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_AEMv8R.bp.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_AEMv8R.bp.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_AEMv8R.bp.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_AEMv8R.bp.vram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_AEMv8R.cci400**

Cache Coherent Interconnect for AXI4 ACE.
Type: `CCI400`.

**FVP_Base_AEMv8R.clockdivider0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_AEMv8R.cluster0**

ARMAEMv8-R Cluster CT model.
Type: `Cluster_ARMAEMv8-R_MP`.

**FVP_Base_AEMv8R.cluster0.cpu0**

ARMAEMv8-R MP CT model.
Type: `ARMAEMv8-R_MP`.

**FVP_Base_AEMv8R.cluster0.cpu0.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_AEMv8R.cluster0.cpu0.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_AEMv8R.cluster0.cpu0.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_AEMv8R.cluster0.cpu1**

ARMAEMv8-R MP CT model.
Type: `ARMAEMv8-R_MP`.

**FVP_Base_AEMv8R.cluster0.cpu1.dtlb**

TLB - instruction, data or unified.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `TlbCadi`.

**FVP_Base_AEMv8R.cluster0.cpu1.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_AEMv8R.cluster0.cpu1.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_AEMv8R.cluster0.cpu2**

ARMAEMv8-R MP CT model.
Type: `ARMAEMv8-R_MP`.

**FVP_Base_AEMv8R.cluster0.cpu2.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_AEMv8R.cluster0.cpu2.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_AEMv8R.cluster0.cpu2.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_AEMv8R.cluster0.cpu3**

ARMAEMv8-R MP CT model.
Type: `ARMAEMv8-R_MP`.

**FVP_Base_AEMv8R.cluster0.cpu3.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_AEMv8R.cluster0.cpu3.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_AEMv8R.cluster0.cpu3.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_AEMv8R.cluster0.l2_cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_AEMv8R.cluster0_labeller**

Type: `Labeller`.

**FVP_Base_AEMv8R.dapmemlogger**

Bus Logger.
Type: `PVBusLogger`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_AEMv8R.elfloader**

ELF loader component.
Type: `ElfLoader`.

**FVP_Base_AEMv8R.gic_distributor**

GIC Interrupt Redistribution Infrastucture component.
Type: `GIC_IRI`.

**FVP_Base_AEMv8R.pctl**

Base Platforms Power Controller.
Type: `Base_PowerController`.

**FVP_Base_AEMv8R.pctl.timer_reset**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_AEMv8R.pctl.timer_reset.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_AEMv8R.pctl.timer_reset.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_AEMv8R.pctl.timer_reset.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

## 13.2 FVP_Base_AEMvA-AEMvA

List of instances in FVP_Base_AEMvA-AEMvA.

### FVP_Base_AEMvA-AEMvA instances

**FVP_Base_AEMvA_AEMvA**

Base Platform Compute Subsystem for AEMvACT and AEMvACT.
Type: `FVP_Base_AEMvA_AEMvA`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_AEMvA_AEMvA.bp**

Peripherals and address map for the Base Platform.
Type: `BasePlatformPeripherals`.

**FVP_Base_AEMvA_AEMvA.bp.Timer_0_1**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_AEMvA_AEMvA.bp.Timer_0_1.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_AEMvA_AEMvA.bp.Timer_0_1.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_AEMvA_AEMvA.bp.Timer_0_1.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_AEMvA_AEMvA.bp.Timer_0_1.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_AEMvA_AEMvA.bp.Timer_2_3**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_AEMvA_AEMvA.bp.Timer_2_3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_AEMvA_AEMvA.bp.Timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_AEMvA_AEMvA.bp.Timer_2_3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_AEMvA_AEMvA.bp.Timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_AEMvA_AEMvA.bp.ap_refclk**

ARM Generic Timer.
Type: `MemoryMappedGenericTimer`.

**FVP_Base_AEMvA_AEMvA.bp.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**FVP_Base_AEMvA_AEMvA.bp.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_AEMvA_AEMvA.bp.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_AEMvA_AEMvA.bp.clock300MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_AEMvA_AEMvA.bp.clock32KHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_AEMvA_AEMvA.bp.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_AEMvA_AEMvA.bp.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_AEMvA_AEMvA.bp.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_AEMvA_AEMvA.bp.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_AEMvA_AEMvA.bp.dmc**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_AEMvA_AEMvA.bp.dmc_phy**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_AEMvA_AEMvA.bp.dummy_local_dap_rom**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_AEMvA_AEMvA.bp.dummy_ram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_AEMvA_AEMvA.bp.dummy_usb**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_AEMvA_AEMvA.bp.exclusive_monitor**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_Base_AEMvA_AEMvA.bp.flash0**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_AEMvA_AEMvA.bp.flash1**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_AEMvA_AEMvA.bp.flashloader0**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_AEMvA_AEMvA.bp.flashloader1**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_AEMvA_AEMvA.bp.generic_watchdog**

ARM Generic Watchdog.
Type: `MemoryMappedGenericWatchdog`.

**FVP_Base_AEMvA_AEMvA.bp.hdlcd0**

ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).
Type: `PL370_HDLCD`.

**FVP_Base_AEMvA_AEMvA.bp.hdlcd0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_AEMvA_AEMvA.bp.hdlcd0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_AEMvA_AEMvA.bp.hdlcd0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_AEMvA_AEMvA.bp.hdlcd0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_AEMvA_AEMvA.bp.hdlcd0_labeller**

Type: `Labeller`.

**FVP_Base_AEMvA_AEMvA.bp.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_Base_AEMvA_AEMvA.bp.mmc**

Generic Multimedia Card.
Type: `MMC`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_AEMvA_AEMvA.bp.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_AEMvA_AEMvA.bp.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_AEMvA_AEMvA.bp.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_AEMvA_AEMvA.bp.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_AEMvA_AEMvA.bp.mpe**

Encrypt memory transactions for each encryption context with an independent key to prevent mismatch access.
Type: `PVMemoryProtectionEngine`.

**FVP_Base_AEMvA_AEMvA.bp.nontrustedrom**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_AEMvA_AEMvA.bp.nontrustedromloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_AEMvA_AEMvA.bp.ns_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_AEMvA_AEMvA.bp.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_AEMvA_AEMvA.bp.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_AEMvA_AEMvA.bp.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_AEMvA_AEMvA.bp.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_AEMvA_AEMvA.bp.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_AEMvA_AEMvA.bp.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_AEMvA_AEMvA.bp.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_AEMvA_AEMvA.bp.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_AEMvA_AEMvA.bp.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ClockTimerThread`.

**FVP_Base_AEMvA_AEMvA.bp.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_AEMvA_AEMvA.bp.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_AEMvA_AEMvA.bp.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_AEMvA_AEMvA.bp.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_AEMvA_AEMvA.bp.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_AEMvA_AEMvA.bp.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_AEMvA_AEMvA.bp.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_AEMvA_AEMvA.bp.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_AEMvA_AEMvA.bp.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_AEMvA_AEMvA.bp.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_AEMvA_AEMvA.bp.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_AEMvA_AEMvA.bp.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_AEMvA_AEMvA.bp.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_AEMvA_AEMvA.bp.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_AEMvA_AEMvA.bp.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_AEMvA_AEMvA.bp.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_AEMvA_AEMvA.bp.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_AEMvA_AEMvA.bp.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_AEMvA_AEMvA.bp.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_AEMvA_AEMvA.bp.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_AEMvA_AEMvA.bp.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_Base_AEMvA_AEMvA.bp.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_AEMvA_AEMvA.bp.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_AEMvA_AEMvA.bp.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_AEMvA_AEMvA.bp.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_AEMvA_AEMvA.bp.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_AEMvA_AEMvA.bp.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_AEMvA_AEMvA.bp.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_AEMvA_AEMvA.bp.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_AEMvA_AEMvA.bp.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_Base_AEMvA_AEMvA.bp.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

**FVP_Base_AEMvA_AEMvA.bp.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_AEMvA_AEMvA.bp.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_AEMvA_AEMvA.bp.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_AEMvA_AEMvA.bp.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_AEMvA_AEMvA.bp.pl111_clcd_labeller**

Type: `Labeller`.

**FVP_Base_AEMvA_AEMvA.bp.pl180_mci**

ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

**FVP_Base_AEMvA_AEMvA.bp.ps2keyboard**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

**FVP_Base_AEMvA_AEMvA.bp.ps2keyboard.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_AEMvA_AEMvA.bp.ps2keyboard.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_AEMvA_AEMvA.bp.ps2keyboard.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_AEMvA_AEMvA.bp.ps2keyboard.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_AEMvA_AEMvA.bp.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

**FVP_Base_AEMvA_AEMvA.bp.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_AEMvA_AEMvA.bp.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_AEMvA_AEMvA.bp.ps2mouse.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_AEMvA_AEMvA.bp.ps2mouse.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_AEMvA_AEMvA.bp.psram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_AEMvA_AEMvA.bp.refcounter**

Memory Mapped Counter Module for Generic Timers.
Type: `MemoryMappedCounterModule`.

**FVP_Base_AEMvA_AEMvA.bp.reset_or**

Or Gate.
Type: `OrGate`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_AEMvA_AEMvA.bp.rl_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_AEMvA_AEMvA.bp.rt_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_AEMvA_AEMvA.bp.s_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_AEMvA_AEMvA.bp.secureDRAM**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_AEMvA_AEMvA.bp.secureSRAM**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_AEMvA_AEMvA.bp.secureflash**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_AEMvA_AEMvA.bp.secureflashloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_AEMvA_AEMvA.bp.smsc_91c111**

SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_Base_AEMvA_AEMvA.bp.sp805_wdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_AEMvA_AEMvA.bp.sp805_wdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_AEMvA_AEMvA.bp.sp805_wdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_AEMvA_AEMvA.bp.sp805_wdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_AEMvA_AEMvA.bp.sp805_wdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_AEMvA_AEMvA.bp.sp810_sysctrl**

Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_Base_AEMvA_AEMvA.bp.sp810_sysctrl.clkdiv_clk0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_AEMvA_AEMvA.bp.sp810_sysctrl.clkdiv_clk1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_AEMvA_AEMvA.bp.sp810_sysctrl.clkdiv_clk2**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_AEMvA_AEMvA.bp.sp810_sysctrl.clkdiv_clk3**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_AEMvA_AEMvA.bp.sram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_AEMvA_AEMvA.bp.terminal_0**

Telnet terminal interface.
Type: `TelnetTerminal`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_AEMvA_AEMvA.bp.terminal_1**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_AEMvA_AEMvA.bp.terminal_2**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_AEMvA_AEMvA.bp.terminal_3**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_AEMvA_AEMvA.bp.trusted_key_storage**

Trusted Root-Key Storage unit.
Type: `RootKeyStorage`.

**FVP_Base_AEMvA_AEMvA.bp.trusted_nv_counter**

Trusted Non-Volatile Counter unit.
Type: `NonVolatileCounter`.

**FVP_Base_AEMvA_AEMvA.bp.trusted_rng**

Random Number Generator unit.
Type: `RandomNumberGenerator`.

**FVP_Base_AEMvA_AEMvA.bp.trusted_watchdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_AEMvA_AEMvA.bp.trusted_watchdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_AEMvA_AEMvA.bp.trusted_watchdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_AEMvA_AEMvA.bp.trusted_watchdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_AEMvA_AEMvA.bp.trusted_watchdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_AEMvA_AEMvA.bp.tzc_400**

TrustZone Address Space Controller.
Type: `TZC_400`.

**FVP_Base_AEMvA_AEMvA.bp.ve_sysregs**

Type: `VE_SysRegs`.

**FVP_Base_AEMvA_AEMvA.bp.vedcc**

Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

**FVP_Base_AEMvA_AEMvA.bp.virtio_net**

VirtioNet device over MMIO transport.
Type: `VirtioNetMMIO`.

**FVP_Base_AEMvA_AEMvA.bp.virtio_net_labeller**

Type: `Labeller`.

**FVP_Base_AEMvA_AEMvA.bp.virtio_rng**

VirtioEntropy device over MMIO transport.
Type: `VirtioEntropyMMIO`.

**FVP_Base_AEMvA_AEMvA.bp.virtioblockdevice**

VirtioBlock device over MMIO transport.
Type: `VirtioBlockMMIO`.

**FVP_Base_AEMvA_AEMvA.bp.virtioblockdevice_labeller**

Type: `Labeller`.

**FVP_Base_AEMvA_AEMvA.bp.virtiop9device**

virtio P9 server.
Type: `VirtioP9Device`.

**FVP_Base_AEMvA_AEMvA.bp.virtiop9device_labeller**

Type: `Labeller`.

**FVP_Base_AEMvA_AEMvA.bp.vis**

Display window for VE using Visualisation library.
Type: `VEVisualisation`.

**FVP_Base_AEMvA_AEMvA.bp.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_Base_AEMvA_AEMvA.bp.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_AEMvA_AEMvA.bp.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_AEMvA_AEMvA.bp.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_AEMvA_AEMvA.bp.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_AEMvA_AEMvA.bp.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_AEMvA_AEMvA.bp.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_AEMvA_AEMvA.bp.vram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_AEMvA_AEMvA.cci400**

Cache Coherent Interconnect for AXI4 ACE.
Type: `CCI400`.

**FVP_Base_AEMvA_AEMvA.clockdivider0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_AEMvA_AEMvA.clockdivider1**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_AEMvA_AEMvA.cluster0**
ARM AEMvA Cluster CT model.
Type: `Cluster_ARM_AEM-A_MP`.

**FVP_Base_AEMvA_AEMvA.cluster0.cpu0**
ARM AEM-A MP CT model.
Type: `ARM_AEM-A_MP`.

**FVP_Base_AEMvA_AEMvA.cluster0.cpu0.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_AEMvA_AEMvA.cluster0.cpu0.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_AEMvA_AEMvA.cluster0.cpu0.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_AEMvA_AEMvA.cluster0.cpu1**
ARM AEM-A MP CT model.
Type: `ARM_AEM-A_MP`.

**FVP_Base_AEMvA_AEMvA.cluster0.cpu1.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_AEMvA_AEMvA.cluster0.cpu1.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_AEMvA_AEMvA.cluster0.cpu1.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_AEMvA_AEMvA.cluster0.cpu2**
ARM AEM-A MP CT model.
Type: `ARM_AEM-A_MP`.

**FVP_Base_AEMvA_AEMvA.cluster0.cpu2.dtlb**
TLB - instruction, data or unified.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `TlbCadi`.

**FVP_Base_AEMvA_AEMvA.cluster0.cpu2.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_AEMvA_AEMvA.cluster0.cpu2.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_AEMvA_AEMvA.cluster0.cpu3**
ARM AEM-A MP CT model.
Type: `ARM_AEM-A_MP`.

**FVP_Base_AEMvA_AEMvA.cluster0.cpu3.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_AEMvA_AEMvA.cluster0.cpu3.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_AEMvA_AEMvA.cluster0.cpu3.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_AEMvA_AEMvA.cluster0.l2_cache**
PV Cache.
Type: `PVCache`.

**FVP_Base_AEMvA_AEMvA.cluster0_labeller**
Type: `Labeller`.

**FVP_Base_AEMvA_AEMvA.cluster1**
ARM AEMvA Cluster CT model.
Type: `Cluster_ARM_AEM-A_MP`.

**FVP_Base_AEMvA_AEMvA.cluster1.cpu0**
ARM AEM-A MP CT model.
Type: `ARM_AEM-A_MP`.

**FVP_Base_AEMvA_AEMvA.cluster1.cpu0.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_AEMvA_AEMvA.cluster1.cpu0.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_AEMvA_AEMvA.cluster1.cpu0.l1icache**
PV Cache.
Type: `PVCache`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_AEMvA_AEMvA.cluster1.cpu1**
ARM AEM-A MP CT model.
Type: `ARM_AEM-A_MP`.

**FVP_Base_AEMvA_AEMvA.cluster1.cpu1.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_AEMvA_AEMvA.cluster1.cpu1.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_AEMvA_AEMvA.cluster1.cpu1.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_AEMvA_AEMvA.cluster1.cpu2**
ARM AEM-A MP CT model.
Type: `ARM_AEM-A_MP`.

**FVP_Base_AEMvA_AEMvA.cluster1.cpu2.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_AEMvA_AEMvA.cluster1.cpu2.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_AEMvA_AEMvA.cluster1.cpu2.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_AEMvA_AEMvA.cluster1.cpu3**
ARM AEM-A MP CT model.
Type: `ARM_AEM-A_MP`.

**FVP_Base_AEMvA_AEMvA.cluster1.cpu3.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_AEMvA_AEMvA.cluster1.cpu3.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_AEMvA_AEMvA.cluster1.cpu3.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_AEMvA_AEMvA.cluster1.l2_cache**
PV Cache.
Type: `PVCache`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_AEMvA_AEMvA.cluster1_labeller**

Type: `Labeller`.

**FVP_Base_AEMvA_AEMvA.dapmemlogger**

Bus Logger.
Type: `PVBusLogger`.

**FVP_Base_AEMvA_AEMvA.elfloader**

ELF loader component.
Type: `ElfLoader`.

**FVP_Base_AEMvA_AEMvA.gic_distributor**

GIC Interrupt Redistribution Infrastucture component.
Type: `GIC_IRI`.

**FVP_Base_AEMvA_AEMvA.pctl**

Base Platforms Power Controller.
Type: `Base_PowerController`.

**FVP_Base_AEMvA_AEMvA.pctl.timer_reset**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_AEMvA_AEMvA.pctl.timer_reset.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_AEMvA_AEMvA.pctl.timer_reset.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_AEMvA_AEMvA.pctl.timer_reset.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

# 13.3 FVP_Base_Cortex-A32x1

List of instances in FVP_Base_Cortex-A32x1.

## FVP_Base_Cortex-A32x1 instances

**FVP_Base_Cortex_A32x1**

Base Platform Compute Subsystem for ARMCortexA32x1CT.
Type: `FVP_Base_Cortex_A32x1`.

**FVP_Base_Cortex_A32x1.bp**

Peripherals and address map for the Base Platform.
Type: `BasePlatformPeripherals`.

**FVP_Base_Cortex_A32x1.bp.Timer_0_1**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Cortex_A32x1.bp.Timer_0_1.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A32x1.bp.Timer_0_1.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A32x1.bp.Timer_0_1.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A32x1.bp.Timer_0_1.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A32x1.bp.Timer_2_3**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Cortex_A32x1.bp.Timer_2_3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A32x1.bp.Timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A32x1.bp.Timer_2_3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A32x1.bp.Timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A32x1.bp.ap_refclk**

ARM Generic Timer.
Type: `MemoryMappedGenericTimer`.

**FVP_Base_Cortex_A32x1.bp.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**FVP_Base_Cortex_A32x1.bp.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A32x1.bp.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A32x1.bp.clock300MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A32x1.bp.clock32KHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A32x1.bp.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A32x1.bp.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A32x1.bp.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A32x1.bp.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A32x1.bp.dmc**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A32x1.bp.dmc_phy**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A32x1.bp.dummy_local_dap_rom**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A32x1.bp.dummy_ram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A32x1.bp.dummy_usb**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A32x1.bp.exclusive_monitor**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A32x1.bp.flash0**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A32x1.bp.flash1**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A32x1.bp.flashloader0**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A32x1.bp.flashloader1**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A32x1.bp.generic_watchdog**

ARM Generic Watchdog.
Type: `MemoryMappedGenericWatchdog`.

**FVP_Base_Cortex_A32x1.bp.hdlcd0**

ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).
Type: `PL370_HDLCD`.

**FVP_Base_Cortex_A32x1.bp.hdlcd0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A32x1.bp.hdlcd0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A32x1.bp.hdlcd0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A32x1.bp.hdlcd0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A32x1.bp.hdlcd0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A32x1.bp.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_Base_Cortex_A32x1.bp.mmc**

Generic Multimedia Card.
Type: `MMC`.

**FVP_Base_Cortex_A32x1.bp.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A32x1.bp.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A32x1.bp.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A32x1.bp.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A32x1.bp.mpe**

Encrypt memory transactions for each encryption context with an independent key to prevent mismatch access.
Type: `PVMemoryProtectionEngine`.

**FVP_Base_Cortex_A32x1.bp.nontrustedrom**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A32x1.bp.nontrustedromloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A32x1.bp.ns_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A32x1.bp.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A32x1.bp.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A32x1.bp.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A32x1.bp.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A32x1.bp.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A32x1.bp.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A32x1.bp.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A32x1.bp.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A32x1.bp.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A32x1.bp.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A32x1.bp.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A32x1.bp.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A32x1.bp.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A32x1.bp.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A32x1.bp.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ClockTimerThread`.

**FVP_Base_Cortex_A32x1.bp.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A32x1.bp.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A32x1.bp.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A32x1.bp.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A32x1.bp.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A32x1.bp.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A32x1.bp.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A32x1.bp.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A32x1.bp.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A32x1.bp.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

**FVP_Base_Cortex_A32x1.bp.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A32x1.bp.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A32x1.bp.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A32x1.bp.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A32x1.bp.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_Base_Cortex_A32x1.bp.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A32x1.bp.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A32x1.bp.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A32x1.bp.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A32x1.bp.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A32x1.bp.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A32x1.bp.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A32x1.bp.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A32x1.bp.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_Base_Cortex_A32x1.bp.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A32x1.bp.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A32x1.bp.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A32x1.bp.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A32x1.bp.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A32x1.bp.pl111_clcd_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A32x1.bp.pl180_mci**

ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

**FVP_Base_Cortex_A32x1.bp.ps2keyboard**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

**FVP_Base_Cortex_A32x1.bp.ps2keyboard.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A32x1.bp.ps2keyboard.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A32x1.bp.ps2keyboard.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A32x1.bp.ps2keyboard.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A32x1.bp.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

**FVP_Base_Cortex_A32x1.bp.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A32x1.bp.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A32x1.bp.ps2mouse.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A32x1.bp.ps2mouse.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A32x1.bp.psram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A32x1.bp.refcounter**
Memory Mapped Counter Module for Generic Timers.
Type: `MemoryMappedCounterModule`.

**FVP_Base_Cortex_A32x1.bp.reset_or**
Or Gate.
Type: `OrGate`.

**FVP_Base_Cortex_A32x1.bp.rl_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A32x1.bp.rt_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A32x1.bp.s_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A32x1.bp.secureDRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A32x1.bp.secureSRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A32x1.bp.secureflash**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A32x1.bp.secureflashloader**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A32x1.bp.smsc_91c111**
SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_Base_Cortex_A32x1.bp.sp805_wdog**
ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A32x1.bp.sp805_wdog.clocktimer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A32x1.bp.sp805_wdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A32x1.bp.sp805_wdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A32x1.bp.sp805_wdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A32x1.bp.sp810_sysctrl**

Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_Base_Cortex_A32x1.bp.sp810_sysctrl.clkdiv_clk0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A32x1.bp.sp810_sysctrl.clkdiv_clk1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A32x1.bp.sp810_sysctrl.clkdiv_clk2**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A32x1.bp.sp810_sysctrl.clkdiv_clk3**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**`FVP_Base_Cortex_A32x1.bp.sram`**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**`FVP_Base_Cortex_A32x1.bp.terminal_0`**
Telnet terminal interface.
Type: `TelnetTerminal`.

**`FVP_Base_Cortex_A32x1.bp.terminal_1`**
Telnet terminal interface.
Type: `TelnetTerminal`.

**`FVP_Base_Cortex_A32x1.bp.terminal_2`**
Telnet terminal interface.
Type: `TelnetTerminal`.

**`FVP_Base_Cortex_A32x1.bp.terminal_3`**
Telnet terminal interface.
Type: `TelnetTerminal`.

**`FVP_Base_Cortex_A32x1.bp.trusted_key_storage`**
Trusted Root-Key Storage unit.
Type: `RootKeyStorage`.

**`FVP_Base_Cortex_A32x1.bp.trusted_nv_counter`**
Trusted Non-Volatile Counter unit.
Type: `NonVolatileCounter`.

**`FVP_Base_Cortex_A32x1.bp.trusted_rng`**
Random Number Generator unit.
Type: `RandomNumberGenerator`.

**`FVP_Base_Cortex_A32x1.bp.trusted_watchdog`**
ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**`FVP_Base_Cortex_A32x1.bp.trusted_watchdog.clocktimer`**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**`FVP_Base_Cortex_A32x1.bp.trusted_watchdog.clocktimer.timer`**
A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A32x1.bp.trusted_watchdog.clocktimer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A32x1.bp.trusted_watchdog.clocktimer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A32x1.bp.tzc_400**
TrustZone Address Space Controller.
Type: `TZC_400`.

**FVP_Base_Cortex_A32x1.bp.ve_sysregs**
Type: `VE_SysRegs`.

**FVP_Base_Cortex_A32x1.bp.vedcc**
Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

**FVP_Base_Cortex_A32x1.bp.virtio_net**
VirtioNet device over MMIO transport.
Type: `VirtioNetMMIO`.

**FVP_Base_Cortex_A32x1.bp.virtio_net_labeller**
Type: `Labeller`.

**FVP_Base_Cortex_A32x1.bp.virtio_rng**
VirtioEntropy device over MMIO transport.
Type: `VirtioEntropyMMIO`.

**FVP_Base_Cortex_A32x1.bp.virtioblockdevice**
VirtioBlock device over MMIO transport.
Type: `VirtioBlockMMIO`.

**FVP_Base_Cortex_A32x1.bp.virtioblockdevice_labeller**
Type: `Labeller`.

**FVP_Base_Cortex_A32x1.bp.virtiop9device**
virtio P9 server.
Type: `VirtioP9Device`.

**FVP_Base_Cortex_A32x1.bp.virtiop9device_labeller**
Type: `Labeller`.

**FVP_Base_Cortex_A32x1.bp.vis**
Display window for VE using Visualisation library.
Type: `VEVisualisation`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A32x1.bp.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_Base_Cortex_A32x1.bp.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A32x1.bp.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A32x1.bp.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A32x1.bp.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A32x1.bp.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A32x1.bp.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A32x1.bp.vram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A32x1.cci400**

Cache Coherent Interconnect for AXI4 ACE.
Type: `CCI400`.

**FVP_Base_Cortex_A32x1.clockdivider0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A32x1.cluster0**

ARM Cortex-A32 Cluster CT model.
Type: `Cluster_ARM_Cortex-A32`.

**FVP_Base_Cortex_A32x1.cluster0.cpu0**

ARM Cortex-A32 CT model.
Type: `ARM_Cortex-A32`.

**FVP_Base_Cortex_A32x1.cluster0.cpu0.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A32x1.cluster0.cpu0.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A32x1.cluster0.cpu0.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A32x1.cluster0.l2_cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A32x1.cluster0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A32x1.dapmemlogger**

Bus Logger.
Type: `PVBusLogger`.

**FVP_Base_Cortex_A32x1.elfloader**

ELF loader component.
Type: `ElfLoader`.

**FVP_Base_Cortex_A32x1.gic_distributor**

GIC Interrupt Redistribution Infrastucture component.
Type: `GIC_IRI`.

**FVP_Base_Cortex_A32x1.pctl**

Base Platforms Power Controller.
Type: `Base_PowerController`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A32x1.pctl.timer_reset**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A32x1.pctl.timer_reset.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A32x1.pctl.timer_reset.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A32x1.pctl.timer_reset.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

## 13.4 FVP_Base_Cortex-A32x2

List of instances in FVP_Base_Cortex-A32x2.

### FVP_Base_Cortex-A32x2 instances

**FVP_Base_Cortex_A32x2**

Base Platform Compute Subsystem for ARMCortexA32x2CT.
Type: `FVP_Base_Cortex_A32x2`.

**FVP_Base_Cortex_A32x2.bp**

Peripherals and address map for the Base Platform.
Type: `BasePlatformPeripherals`.

**FVP_Base_Cortex_A32x2.bp.Timer_0_1**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Cortex_A32x2.bp.Timer_0_1.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**`FVP_Base_Cortex_A32x2.bp.Timer_0_1.clk_div1`**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**`FVP_Base_Cortex_A32x2.bp.Timer_0_1.counter0`**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**`FVP_Base_Cortex_A32x2.bp.Timer_0_1.counter1`**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**`FVP_Base_Cortex_A32x2.bp.Timer_2_3`**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**`FVP_Base_Cortex_A32x2.bp.Timer_2_3.clk_div0`**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**`FVP_Base_Cortex_A32x2.bp.Timer_2_3.clk_div1`**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**`FVP_Base_Cortex_A32x2.bp.Timer_2_3.counter0`**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**`FVP_Base_Cortex_A32x2.bp.Timer_2_3.counter1`**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**`FVP_Base_Cortex_A32x2.bp.ap_refclk`**

ARM Generic Timer.
Type: `MemoryMappedGenericTimer`.

**`FVP_Base_Cortex_A32x2.bp.audioout`**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A32x2.bp.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A32x2.bp.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A32x2.bp.clock300MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A32x2.bp.clock32KHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A32x2.bp.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A32x2.bp.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A32x2.bp.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A32x2.bp.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A32x2.bp.dmc**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A32x2.bp.dmc_phy**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A32x2.bp.dummy_local_dap_rom**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A32x2.bp.dummy_ram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A32x2.bp.dummy_usb**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A32x2.bp.exclusive_monitor**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_Base_Cortex_A32x2.bp.flash0**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A32x2.bp.flash1**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A32x2.bp.flashloader0**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A32x2.bp.flashloader1**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A32x2.bp.generic_watchdog**

ARM Generic Watchdog.
Type: `MemoryMappedGenericWatchdog`.

**FVP_Base_Cortex_A32x2.bp.hdlcd0**

ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `PL370_HDLCD`.

**FVP_Base_Cortex_A32x2.bp.hdlcd0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A32x2.bp.hdlcd0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A32x2.bp.hdlcd0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A32x2.bp.hdlcd0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A32x2.bp.hdlcd0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A32x2.bp.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_Base_Cortex_A32x2.bp.mmc**

Generic Multimedia Card.
Type: `MMC`.

**FVP_Base_Cortex_A32x2.bp.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A32x2.bp.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A32x2.bp.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A32x2.bp.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A32x2.bp.mpe**

Encrypt memory transactions for each encryption context with an independent key to prevent mismatch access.
Type: `PVMemoryProtectionEngine`.

**FVP_Base_Cortex_A32x2.bp.nontrustedrom**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A32x2.bp.nontrustedromloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A32x2.bp.ns_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A32x2.bp.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A32x2.bp.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A32x2.bp.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A32x2.bp.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A32x2.bp.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A32x2.bp.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A32x2.bp.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A32x2.bp.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A32x2.bp.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A32x2.bp.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A32x2.bp.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A32x2.bp.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A32x2.bp.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A32x2.bp.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A32x2.bp.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A32x2.bp.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A32x2.bp.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A32x2.bp.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A32x2.bp.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A32x2.bp.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A32x2.bp.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A32x2.bp.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A32x2.bp.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A32x2.bp.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A32x2.bp.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

**FVP_Base_Cortex_A32x2.bp.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A32x2.bp.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A32x2.bp.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A32x2.bp.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A32x2.bp.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_Base_Cortex_A32x2.bp.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A32x2.bp.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A32x2.bp.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A32x2.bp.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A32x2.bp.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A32x2.bp.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A32x2.bp.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A32x2.bp.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A32x2.bp.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_Base_Cortex_A32x2.bp.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

**FVP_Base_Cortex_A32x2.bp.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A32x2.bp.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A32x2.bp.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `SchedulerThread`.

**FVP_Base_Cortex_A32x2.bp.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A32x2.bp.pl111_clcd_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A32x2.bp.pl180_mci**

ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

**FVP_Base_Cortex_A32x2.bp.ps2keyboard**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

**FVP_Base_Cortex_A32x2.bp.ps2keyboard.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A32x2.bp.ps2keyboard.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A32x2.bp.ps2keyboard.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A32x2.bp.ps2keyboard.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A32x2.bp.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `PS2Mouse`.

**FVP_Base_Cortex_A32x2.bp.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A32x2.bp.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A32x2.bp.ps2mouse.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A32x2.bp.ps2mouse.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A32x2.bp.psram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A32x2.bp.refcounter**

Memory Mapped Counter Module for Generic Timers.
Type: `MemoryMappedCounterModule`.

**FVP_Base_Cortex_A32x2.bp.reset_or**

Or Gate.
Type: `OrGate`.

**FVP_Base_Cortex_A32x2.bp.rl_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A32x2.bp.rt_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A32x2.bp.s_dram**

RAM device, can be dynamic or static ram.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `RAMDevice`.

**FVP_Base_Cortex_A32x2.bp.secureDRAM**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A32x2.bp.secureSRAM**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A32x2.bp.secureflash**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A32x2.bp.secureflashloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A32x2.bp.smsc_91c111**

SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_Base_Cortex_A32x2.bp.sp805_wdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A32x2.bp.sp805_wdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A32x2.bp.sp805_wdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A32x2.bp.sp805_wdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A32x2.bp.sp805_wdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A32x2.bp.sp810_sysctrl**

Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_Base_Cortex_A32x2.bp.sp810_sysctrl.clkdiv_clk0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A32x2.bp.sp810_sysctrl.clkdiv_clk1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A32x2.bp.sp810_sysctrl.clkdiv_clk2**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A32x2.bp.sp810_sysctrl.clkdiv_clk3**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A32x2.bp.sram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A32x2.bp.terminal_0**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A32x2.bp.terminal_1**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A32x2.bp.terminal_2**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A32x2.bp.terminal_3**

Telnet terminal interface.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `TelnetTerminal`.

**FVP_Base_Cortex_A32x2.bp.trusted_key_storage**

Trusted Root-Key Storage unit.
Type: `RootKeyStorage`.

**FVP_Base_Cortex_A32x2.bp.trusted_nv_counter**

Trusted Non-Volatile Counter unit.
Type: `NonVolatileCounter`.

**FVP_Base_Cortex_A32x2.bp.trusted_rng**

Random Number Generator unit.
Type: `RandomNumberGenerator`.

**FVP_Base_Cortex_A32x2.bp.trusted_watchdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A32x2.bp.trusted_watchdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A32x2.bp.trusted_watchdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A32x2.bp.trusted_watchdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A32x2.bp.trusted_watchdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A32x2.bp.tzc_400**

TrustZone Address Space Controller.
Type: `TZC_400`.

**FVP_Base_Cortex_A32x2.bp.ve_sysregs**

Type: `VE_SysRegs`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A32x2.bp.vedcc**

Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

**FVP_Base_Cortex_A32x2.bp.virtio_net**

VirtioNet device over MMIO transport.
Type: `VirtioNetMMIO`.

**FVP_Base_Cortex_A32x2.bp.virtio_net_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A32x2.bp.virtio_rng**

VirtioEntropy device over MMIO transport.
Type: `VirtioEntropyMMIO`.

**FVP_Base_Cortex_A32x2.bp.virtioblockdevice**

VirtioBlock device over MMIO transport.
Type: `VirtioBlockMMIO`.

**FVP_Base_Cortex_A32x2.bp.virtioblockdevice_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A32x2.bp.virtiop9device**

virtio P9 server.
Type: `VirtioP9Device`.

**FVP_Base_Cortex_A32x2.bp.virtiop9device_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A32x2.bp.vis**

Display window for VE using Visualisation library.
Type: `VEVisualisation`.

**FVP_Base_Cortex_A32x2.bp.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_Base_Cortex_A32x2.bp.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A32x2.bp.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ClockTimerThread`.

**FVP_Base_Cortex_A32x2.bp.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A32x2.bp.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A32x2.bp.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A32x2.bp.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A32x2.bp.vram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A32x2.cci400**

Cache Coherent Interconnect for AXI4 ACE.
Type: `CCI400`.

**FVP_Base_Cortex_A32x2.clockdivider0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A32x2.cluster0**

ARM Cortex-A32 Cluster CT model.
Type: `Cluster_ARM_Cortex-A32`.

**FVP_Base_Cortex_A32x2.cluster0.cpu0**

ARM Cortex-A32 CT model.
Type: `ARM_Cortex-A32`.

**FVP_Base_Cortex_A32x2.cluster0.cpu0.dtlb**

TLB - instruction, data or unified.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `TlbCadi`.

**FVP_Base_Cortex_A32x2.cluster0.cpu0.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A32x2.cluster0.cpu0.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A32x2.cluster0.cpu1**

ARM Cortex-A32 CT model.
Type: `ARM_Cortex-A32`.

**FVP_Base_Cortex_A32x2.cluster0.cpu1.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A32x2.cluster0.cpu1.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A32x2.cluster0.cpu1.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A32x2.cluster0.l2_cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A32x2.cluster0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A32x2.dapmemlogger**

Bus Logger.
Type: `PVBusLogger`.

**FVP_Base_Cortex_A32x2.elfloader**

ELF loader component.
Type: `ElfLoader`.

**FVP_Base_Cortex_A32x2.gic_distributor**

GIC Interrupt Redistribution Infrastucture component.
Type: `GIC_IRI`.

**FVP_Base_Cortex_A32x2.pctl**

Base Platforms Power Controller.
Type: `Base_PowerController`.

**FVP_Base_Cortex_A32x2.pctl.timer_reset**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**`FVP_Base_Cortex_A32x2.pctl.timer_reset.timer`**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**`FVP_Base_Cortex_A32x2.pctl.timer_reset.timer.thread`**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**`FVP_Base_Cortex_A32x2.pctl.timer_reset.timer.thread_event`**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

## 13.5 FVP_Base_Cortex-A32x4

List of instances in FVP_Base_Cortex-A32x4.

### FVP_Base_Cortex-A32x4 instances

**`FVP_Base_Cortex_A32x4`**

Base Platform Compute Subsystem for ARMCortexA32x4CT.
Type: `FVP_Base_Cortex_A32x4`.

**`FVP_Base_Cortex_A32x4.bp`**

Peripherals and address map for the Base Platform.
Type: `BasePlatformPeripherals`.

**`FVP_Base_Cortex_A32x4.bp.Timer_0_1`**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**`FVP_Base_Cortex_A32x4.bp.Timer_0_1.clk_div0`**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A32x4.bp.Timer_0_1.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A32x4.bp.Timer_0_1.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A32x4.bp.Timer_0_1.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A32x4.bp.Timer_2_3**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Cortex_A32x4.bp.Timer_2_3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A32x4.bp.Timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A32x4.bp.Timer_2_3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A32x4.bp.Timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A32x4.bp.ap_refclk**

ARM Generic Timer.
Type: `MemoryMappedGenericTimer`.

**FVP_Base_Cortex_A32x4.bp.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**FVP_Base_Cortex_A32x4.bp.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A32x4.bp.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A32x4.bp.clock300MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A32x4.bp.clock32KHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A32x4.bp.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A32x4.bp.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A32x4.bp.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A32x4.bp.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ClockDivider`.

**FVP_Base_Cortex_A32x4.bp.dmc**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A32x4.bp.dmc_phy**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A32x4.bp.dummy_local_dap_rom**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A32x4.bp.dummy_ram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A32x4.bp.dummy_usb**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A32x4.bp.exclusive_monitor**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_Base_Cortex_A32x4.bp.flash0**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A32x4.bp.flash1**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A32x4.bp.flashloader0**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A32x4.bp.flashloader1**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A32x4.bp.generic_watchdog**

ARM Generic Watchdog.
Type: `MemoryMappedGenericWatchdog`.

**FVP_Base_Cortex_A32x4.bp.hdlcd0**

ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).
Type: `PL370_HDLCD`.

**FVP_Base_Cortex_A32x4.bp.hdlcd0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A32x4.bp.hdlcd0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A32x4.bp.hdlcd0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A32x4.bp.hdlcd0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A32x4.bp.hdlcd0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A32x4.bp.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_Base_Cortex_A32x4.bp.mmc**

Generic Multimedia Card.
Type: `MMC`.

**FVP_Base_Cortex_A32x4.bp.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A32x4.bp.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A32x4.bp.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A32x4.bp.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A32x4.bp.mpe**

Encrypt memory transactions for each encryption context with an independent key to prevent mismatch access.
Type: `PVMemoryProtectionEngine`.

**FVP_Base_Cortex_A32x4.bp.nontrustedrom**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A32x4.bp.nontrustedromloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A32x4.bp.ns_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A32x4.bp.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A32x4.bp.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A32x4.bp.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A32x4.bp.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A32x4.bp.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A32x4.bp.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A32x4.bp.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A32x4.bp.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A32x4.bp.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A32x4.bp.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A32x4.bp.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A32x4.bp.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A32x4.bp.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A32x4.bp.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A32x4.bp.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A32x4.bp.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A32x4.bp.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A32x4.bp.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A32x4.bp.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A32x4.bp.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A32x4.bp.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A32x4.bp.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A32x4.bp.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A32x4.bp.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A32x4.bp.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

**FVP_Base_Cortex_A32x4.bp.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A32x4.bp.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A32x4.bp.pl031_rtc.timer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A32x4.bp.pl031_rtc.timer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A32x4.bp.pl041_aaci**
ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_Base_Cortex_A32x4.bp.pl041_aaci.timer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A32x4.bp.pl041_aaci.timer.timer**
A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A32x4.bp.pl041_aaci.timer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A32x4.bp.pl041_aaci.timer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A32x4.bp.pl050_kmi0**
ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A32x4.bp.pl050_kmi0.clk_divider**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**`FVP_Base_Cortex_A32x4.bp.pl050_kmi1`**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**`FVP_Base_Cortex_A32x4.bp.pl050_kmi1.clk_divider`**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**`FVP_Base_Cortex_A32x4.bp.pl111_clcd`**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**`FVP_Base_Cortex_A32x4.bp.pl111_clcd.pl11x_clcd`**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

**`FVP_Base_Cortex_A32x4.bp.pl111_clcd.pl11x_clcd.timer`**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**`FVP_Base_Cortex_A32x4.bp.pl111_clcd.pl11x_clcd.timer.timer`**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**`FVP_Base_Cortex_A32x4.bp.pl111_clcd.pl11x_clcd.timer.timer.thread`**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**`FVP_Base_Cortex_A32x4.bp.pl111_clcd.pl11x_clcd.timer.timer.thread_event`**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A32x4.bp.pl111_clcd_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A32x4.bp.pl180_mci**

ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

**FVP_Base_Cortex_A32x4.bp.ps2keyboard**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

**FVP_Base_Cortex_A32x4.bp.ps2keyboard.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A32x4.bp.ps2keyboard.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A32x4.bp.ps2keyboard.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A32x4.bp.ps2keyboard.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A32x4.bp.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

**FVP_Base_Cortex_A32x4.bp.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A32x4.bp.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A32x4.bp.ps2mouse.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A32x4.bp.ps2mouse.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A32x4.bp.psram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A32x4.bp.refcounter**

Memory Mapped Counter Module for Generic Timers.
Type: `MemoryMappedCounterModule`.

**FVP_Base_Cortex_A32x4.bp.reset_or**

Or Gate.
Type: `OrGate`.

**FVP_Base_Cortex_A32x4.bp.rl_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A32x4.bp.rt_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A32x4.bp.s_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A32x4.bp.secureDRAM**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A32x4.bp.secureSRAM**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A32x4.bp.secureflash**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A32x4.bp.secureflashloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A32x4.bp.smsc_91c111**

SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_Base_Cortex_A32x4.bp.sp805_wdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A32x4.bp.sp805_wdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A32x4.bp.sp805_wdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A32x4.bp.sp805_wdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A32x4.bp.sp805_wdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A32x4.bp.sp810_sysctrl**

Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A32x4.bp.sp810_sysctrl.clkdiv_clk0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A32x4.bp.sp810_sysctrl.clkdiv_clk1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A32x4.bp.sp810_sysctrl.clkdiv_clk2**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A32x4.bp.sp810_sysctrl.clkdiv_clk3**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A32x4.bp.sram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A32x4.bp.terminal_0**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A32x4.bp.terminal_1**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A32x4.bp.terminal_2**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A32x4.bp.terminal_3**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A32x4.bp.trusted_key_storage**

Trusted Root-Key Storage unit.
Type: `RootKeyStorage`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A32x4.bp.trusted_nv_counter**

Trusted Non-Volatile Counter unit.
Type: `NonVolatileCounter`.

**FVP_Base_Cortex_A32x4.bp.trusted_rng**

Random Number Generator unit.
Type: `RandomNumberGenerator`.

**FVP_Base_Cortex_A32x4.bp.trusted_watchdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A32x4.bp.trusted_watchdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A32x4.bp.trusted_watchdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A32x4.bp.trusted_watchdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A32x4.bp.trusted_watchdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A32x4.bp.tzc_400**

TrustZone Address Space Controller.
Type: `TZC_400`.

**FVP_Base_Cortex_A32x4.bp.ve_sysregs**

Type: `VE_SysRegs`.

**FVP_Base_Cortex_A32x4.bp.vedcc**

Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A32x4.bp.virtio_net**

VirtioNet device over MMIO transport.
Type: `VirtioNetMMIO`.

**FVP_Base_Cortex_A32x4.bp.virtio_net_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A32x4.bp.virtio_rng**

VirtioEntropy device over MMIO transport.
Type: `VirtioEntropyMMIO`.

**FVP_Base_Cortex_A32x4.bp.virtioblockdevice**

VirtioBlock device over MMIO transport.
Type: `VirtioBlockMMIO`.

**FVP_Base_Cortex_A32x4.bp.virtioblockdevice_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A32x4.bp.virtiop9device**

virtio P9 server.
Type: `VirtioP9Device`.

**FVP_Base_Cortex_A32x4.bp.virtiop9device_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A32x4.bp.vis**

Display window for VE using Visualisation library.
Type: `VEVisualisation`.

**FVP_Base_Cortex_A32x4.bp.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_Base_Cortex_A32x4.bp.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A32x4.bp.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A32x4.bp.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A32x4.bp.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A32x4.bp.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A32x4.bp.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A32x4.bp.vram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A32x4.cci400**

Cache Coherent Interconnect for AXI4 ACE.
Type: `CCI400`.

**FVP_Base_Cortex_A32x4.clockdivider0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A32x4.cluster0**

ARM Cortex-A32 Cluster CT model.
Type: `Cluster_ARM_Cortex-A32`.

**FVP_Base_Cortex_A32x4.cluster0.cpu0**

ARM Cortex-A32 CT model.
Type: `ARM_Cortex-A32`.

**FVP_Base_Cortex_A32x4.cluster0.cpu0.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A32x4.cluster0.cpu0.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A32x4.cluster0.cpu0.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A32x4.cluster0.cpu1**
ARM Cortex-A32 CT model.
Type: `ARM_Cortex-A32`.

**FVP_Base_Cortex_A32x4.cluster0.cpu1.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A32x4.cluster0.cpu1.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A32x4.cluster0.cpu1.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A32x4.cluster0.cpu2**
ARM Cortex-A32 CT model.
Type: `ARM_Cortex-A32`.

**FVP_Base_Cortex_A32x4.cluster0.cpu2.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A32x4.cluster0.cpu2.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A32x4.cluster0.cpu2.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A32x4.cluster0.cpu3**
ARM Cortex-A32 CT model.
Type: `ARM_Cortex-A32`.

**FVP_Base_Cortex_A32x4.cluster0.cpu3.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A32x4.cluster0.cpu3.l1dcache**
PV Cache.
Type: `PVCache`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A32x4.cluster0.cpu3.l1icache**

PV Cache.
Type: PVCache.

**FVP_Base_Cortex_A32x4.cluster0.l2_cache**

PV Cache.
Type: PVCache.

**FVP_Base_Cortex_A32x4.cluster0_labeller**

Type: Labeller.

**FVP_Base_Cortex_A32x4.dapmemlogger**

Bus Logger.
Type: PVBusLogger.

**FVP_Base_Cortex_A32x4.elfloader**

ELF loader component.
Type: ElfLoader.

**FVP_Base_Cortex_A32x4.gic_distributor**

GIC Interrupt Redistribution Infrastucture component.
Type: GIC_IRI.

**FVP_Base_Cortex_A32x4.pctl**

Base Platforms Power Controller.
Type: Base_PowerController.

**FVP_Base_Cortex_A32x4.pctl.timer_reset**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: ClockTimerThread.

**FVP_Base_Cortex_A32x4.pctl.timer_reset.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: ClockTimerThread64.

**FVP_Base_Cortex_A32x4.pctl.timer_reset.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: SchedulerThread.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A32x4.pctl.timer_reset.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

# 13.6 FVP_Base_Cortex-A34x1

List of instances in FVP_Base_Cortex-A34x1.

## FVP_Base_Cortex-A34x1 instances

**FVP_Base_Cortex_A34x1**

Base Platform Compute Subsystem for ARMCortexA34x1CT.
Type: `FVP_Base_Cortex_A34x1`.

**FVP_Base_Cortex_A34x1.bp**

Peripherals and address map for the Base Platform.
Type: `BasePlatformPeripherals`.

**FVP_Base_Cortex_A34x1.bp.Timer_0_1**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Cortex_A34x1.bp.Timer_0_1.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A34x1.bp.Timer_0_1.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A34x1.bp.Timer_0_1.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A34x1.bp.Timer_0_1.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A34x1.bp.Timer_2_3**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A34x1.bp.Timer_2_3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A34x1.bp.Timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A34x1.bp.Timer_2_3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A34x1.bp.Timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A34x1.bp.ap_refclk**

ARM Generic Timer.
Type: `MemoryMappedGenericTimer`.

**FVP_Base_Cortex_A34x1.bp.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**FVP_Base_Cortex_A34x1.bp.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A34x1.bp.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A34x1.bp.clock300MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A34x1.bp.clock32KHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A34x1.bp.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A34x1.bp.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A34x1.bp.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A34x1.bp.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A34x1.bp.dmc**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A34x1.bp.dmc_phy**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A34x1.bp.dummy_local_dap_rom**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A34x1.bp.dummy_ram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A34x1.bp.dummy_usb**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A34x1.bp.exclusive_monitor**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_Base_Cortex_A34x1.bp.flash0**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A34x1.bp.flash1**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A34x1.bp.flashloader0**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A34x1.bp.flashloader1**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A34x1.bp.generic_watchdog**

ARM Generic Watchdog.
Type: `MemoryMappedGenericWatchdog`.

**FVP_Base_Cortex_A34x1.bp.hdlcd0**

ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).
Type: `PL370_HDLCD`.

**FVP_Base_Cortex_A34x1.bp.hdlcd0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A34x1.bp.hdlcd0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A34x1.bp.hdlcd0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A34x1.bp.hdlcd0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A34x1.bp.hdlcd0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A34x1.bp.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_Base_Cortex_A34x1.bp.mmc**

Generic Multimedia Card.
Type: `MMC`.

**FVP_Base_Cortex_A34x1.bp.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A34x1.bp.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A34x1.bp.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A34x1.bp.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A34x1.bp.mpe**

Encrypt memory transactions for each encryption context with an independent key to prevent mismatch access.
Type: `PVMemoryProtectionEngine`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A34x1.bp.nontrustedrom**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A34x1.bp.nontrustedromloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A34x1.bp.ns_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A34x1.bp.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A34x1.bp.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A34x1.bp.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A34x1.bp.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A34x1.bp.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A34x1.bp.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A34x1.bp.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A34x1.bp.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A34x1.bp.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A34x1.bp.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A34x1.bp.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A34x1.bp.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A34x1.bp.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A34x1.bp.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A34x1.bp.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A34x1.bp.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A34x1.bp.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A34x1.bp.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A34x1.bp.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A34x1.bp.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A34x1.bp.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A34x1.bp.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A34x1.bp.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A34x1.bp.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A34x1.bp.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

**FVP_Base_Cortex_A34x1.bp.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A34x1.bp.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A34x1.bp.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A34x1.bp.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A34x1.bp.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_Base_Cortex_A34x1.bp.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A34x1.bp.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A34x1.bp.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A34x1.bp.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A34x1.bp.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A34x1.bp.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A34x1.bp.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A34x1.bp.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A34x1.bp.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_Base_Cortex_A34x1.bp.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

**FVP_Base_Cortex_A34x1.bp.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A34x1.bp.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A34x1.bp.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A34x1.bp.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A34x1.bp.pl111_clcd_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A34x1.bp.pl180_mci**

ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

**FVP_Base_Cortex_A34x1.bp.ps2keyboard**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A34x1.bp.ps2keyboard.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A34x1.bp.ps2keyboard.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A34x1.bp.ps2keyboard.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A34x1.bp.ps2keyboard.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A34x1.bp.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

**FVP_Base_Cortex_A34x1.bp.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A34x1.bp.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A34x1.bp.ps2mouse.ps2_clocktimer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A34x1.bp.ps2mouse.ps2_clocktimer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A34x1.bp.psram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A34x1.bp.refcounter**
Memory Mapped Counter Module for Generic Timers.
Type: `MemoryMappedCounterModule`.

**FVP_Base_Cortex_A34x1.bp.reset_or**
Or Gate.
Type: `OrGate`.

**FVP_Base_Cortex_A34x1.bp.rl_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A34x1.bp.rt_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A34x1.bp.s_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A34x1.bp.secureDRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A34x1.bp.secureSRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A34x1.bp.secureflash**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A34x1.bp.secureflashloader**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A34x1.bp.smsc_91c111**

SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_Base_Cortex_A34x1.bp.sp805_wdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A34x1.bp.sp805_wdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A34x1.bp.sp805_wdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A34x1.bp.sp805_wdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A34x1.bp.sp805_wdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A34x1.bp.sp810_sysctrl**

Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_Base_Cortex_A34x1.bp.sp810_sysctrl.clkdiv_clk0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A34x1.bp.sp810_sysctrl.clkdiv_clk1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A34x1.bp.sp810_sysctrl.clkdiv_clk2**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A34x1.bp.sp810_sysctrl.clkdiv_clk3**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A34x1.bp.sram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A34x1.bp.terminal_0**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A34x1.bp.terminal_1**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A34x1.bp.terminal_2**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A34x1.bp.terminal_3**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A34x1.bp.trusted_key_storage**
Trusted Root-Key Storage unit.
Type: `RootKeyStorage`.

**FVP_Base_Cortex_A34x1.bp.trusted_nv_counter**
Trusted Non-Volatile Counter unit.
Type: `NonVolatileCounter`.

**FVP_Base_Cortex_A34x1.bp.trusted_rng**
Random Number Generator unit.
Type: `RandomNumberGenerator`.

**FVP_Base_Cortex_A34x1.bp.trusted_watchdog**
ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A34x1.bp.trusted_watchdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A34x1.bp.trusted_watchdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A34x1.bp.trusted_watchdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A34x1.bp.trusted_watchdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A34x1.bp.tzc_400**

TrustZone Address Space Controller.
Type: `TZC_400`.

**FVP_Base_Cortex_A34x1.bp.ve_sysregs**

Type: `VE_SysRegs`.

**FVP_Base_Cortex_A34x1.bp.vedcc**

Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

**FVP_Base_Cortex_A34x1.bp.virtio_net**

VirtioNet device over MMIO transport.
Type: `VirtioNetMMIO`.

**FVP_Base_Cortex_A34x1.bp.virtio_net_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A34x1.bp.virtio_rng**

VirtioEntropy device over MMIO transport.
Type: `VirtioEntropyMMIO`.

**FVP_Base_Cortex_A34x1.bp.virtioblockdevice**

VirtioBlock device over MMIO transport.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `VirtioBlockMMIO`.

**FVP_Base_Cortex_A34x1.bp.virtioblockdevice_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A34x1.bp.virtiop9device**

virtio P9 server.
Type: `VirtioP9Device`.

**FVP_Base_Cortex_A34x1.bp.virtiop9device_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A34x1.bp.vis**

Display window for VE using Visualisation library.
Type: `VEVisualisation`.

**FVP_Base_Cortex_A34x1.bp.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_Base_Cortex_A34x1.bp.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A34x1.bp.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A34x1.bp.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A34x1.bp.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A34x1.bp.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A34x1.bp.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A34x1.bp.vram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A34x1.cci400**

Cache Coherent Interconnect for AXI4 ACE.
Type: `CCI400`.

**FVP_Base_Cortex_A34x1.clockdivider0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A34x1.cluster0**

ARM Cortex-A34 Cluster CT model.
Type: `Cluster_ARM_Cortex-A34`.

**FVP_Base_Cortex_A34x1.cluster0.cpu0**

ARM Cortex-A34 CT model.
Type: `ARM_Cortex-A34`.

**FVP_Base_Cortex_A34x1.cluster0.cpu0.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A34x1.cluster0.cpu0.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A34x1.cluster0.cpu0.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A34x1.cluster0.l2_cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A34x1.cluster0_labeller**

Type: `Labeller`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A34x1.dapmemlogger**

Bus Logger.
Type: `PVBusLogger`.

**FVP_Base_Cortex_A34x1.elfloader**

ELF loader component.
Type: `ElfLoader`.

**FVP_Base_Cortex_A34x1.gic_distributor**

GIC Interrupt Redistribution Infrastucture component.
Type: `GIC_IRI`.

**FVP_Base_Cortex_A34x1.pctl**

Base Platforms Power Controller.
Type: `Base_PowerController`.

**FVP_Base_Cortex_A34x1.pctl.timer_reset**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A34x1.pctl.timer_reset.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A34x1.pctl.timer_reset.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A34x1.pctl.timer_reset.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

# 13.7 FVP_Base_Cortex-A34x2

List of instances in FVP_Base_Cortex-A34x2.

## FVP_Base_Cortex-A34x2 instances

**FVP_Base_Cortex_A34x2**

Base Platform Compute Subsystem for ARMCortexA34x2CT.
Type: `FVP_Base_Cortex_A34x2`.

**FVP_Base_Cortex_A34x2.bp**

Peripherals and address map for the Base Platform.
Type: `BasePlatformPeripherals`.

**FVP_Base_Cortex_A34x2.bp.Timer_0_1**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Cortex_A34x2.bp.Timer_0_1.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A34x2.bp.Timer_0_1.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A34x2.bp.Timer_0_1.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A34x2.bp.Timer_0_1.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A34x2.bp.Timer_2_3**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Cortex_A34x2.bp.Timer_2_3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A34x2.bp.Timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A34x2.bp.Timer_2_3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A34x2.bp.Timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A34x2.bp.ap_refclk**

ARM Generic Timer.
Type: `MemoryMappedGenericTimer`.

**FVP_Base_Cortex_A34x2.bp.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**FVP_Base_Cortex_A34x2.bp.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A34x2.bp.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A34x2.bp.clock300MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A34x2.bp.clock32KHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A34x2.bp.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A34x2.bp.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A34x2.bp.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A34x2.bp.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A34x2.bp.dmc**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A34x2.bp.dmc_phy**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A34x2.bp.dummy_local_dap_rom**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A34x2.bp.dummy_ram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A34x2.bp.dummy_usb**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A34x2.bp.exclusive_monitor**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A34x2.bp.flash0**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A34x2.bp.flash1**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A34x2.bp.flashloader0**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A34x2.bp.flashloader1**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A34x2.bp.generic_watchdog**

ARM Generic Watchdog.
Type: `MemoryMappedGenericWatchdog`.

**FVP_Base_Cortex_A34x2.bp.hdlcd0**

ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).
Type: `PL370_HDLCD`.

**FVP_Base_Cortex_A34x2.bp.hdlcd0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A34x2.bp.hdlcd0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A34x2.bp.hdlcd0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A34x2.bp.hdlcd0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A34x2.bp.hdlcd0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A34x2.bp.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_Base_Cortex_A34x2.bp.mmc**

Generic Multimedia Card.
Type: `MMC`.

**FVP_Base_Cortex_A34x2.bp.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A34x2.bp.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A34x2.bp.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A34x2.bp.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A34x2.bp.mpe**

Encrypt memory transactions for each encryption context with an independent key to prevent mismatch access.
Type: `PVMemoryProtectionEngine`.

**FVP_Base_Cortex_A34x2.bp.nontrustedrom**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A34x2.bp.nontrustedromloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A34x2.bp.ns_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A34x2.bp.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A34x2.bp.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A34x2.bp.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A34x2.bp.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A34x2.bp.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A34x2.bp.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A34x2.bp.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A34x2.bp.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A34x2.bp.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A34x2.bp.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A34x2.bp.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A34x2.bp.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A34x2.bp.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A34x2.bp.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A34x2.bp.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ClockTimerThread`.

**FVP_Base_Cortex_A34x2.bp.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A34x2.bp.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A34x2.bp.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A34x2.bp.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A34x2.bp.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A34x2.bp.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A34x2.bp.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A34x2.bp.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A34x2.bp.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A34x2.bp.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

**FVP_Base_Cortex_A34x2.bp.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A34x2.bp.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A34x2.bp.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A34x2.bp.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A34x2.bp.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_Base_Cortex_A34x2.bp.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A34x2.bp.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A34x2.bp.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A34x2.bp.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A34x2.bp.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A34x2.bp.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A34x2.bp.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A34x2.bp.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A34x2.bp.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_Base_Cortex_A34x2.bp.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A34x2.bp.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A34x2.bp.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A34x2.bp.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A34x2.bp.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A34x2.bp.pl111_clcd_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A34x2.bp.pl180_mci**

ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

**FVP_Base_Cortex_A34x2.bp.ps2keyboard**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

**FVP_Base_Cortex_A34x2.bp.ps2keyboard.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A34x2.bp.ps2keyboard.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A34x2.bp.ps2keyboard.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A34x2.bp.ps2keyboard.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A34x2.bp.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

**FVP_Base_Cortex_A34x2.bp.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A34x2.bp.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A34x2.bp.ps2mouse.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A34x2.bp.ps2mouse.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A34x2.bp.psram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A34x2.bp.refcounter**
Memory Mapped Counter Module for Generic Timers.
Type: `MemoryMappedCounterModule`.

**FVP_Base_Cortex_A34x2.bp.reset_or**
Or Gate.
Type: `OrGate`.

**FVP_Base_Cortex_A34x2.bp.rl_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A34x2.bp.rt_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A34x2.bp.s_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A34x2.bp.secureDRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A34x2.bp.secureSRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A34x2.bp.secureflash**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A34x2.bp.secureflashloader**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A34x2.bp.smsc_91c111**
SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_Base_Cortex_A34x2.bp.sp805_wdog**
ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A34x2.bp.sp805_wdog.clocktimer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A34x2.bp.sp805_wdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A34x2.bp.sp805_wdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A34x2.bp.sp805_wdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A34x2.bp.sp810_sysctrl**

Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_Base_Cortex_A34x2.bp.sp810_sysctrl.clkdiv_clk0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A34x2.bp.sp810_sysctrl.clkdiv_clk1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A34x2.bp.sp810_sysctrl.clkdiv_clk2**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A34x2.bp.sp810_sysctrl.clkdiv_clk3**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A34x2.bp.sram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A34x2.bp.terminal_0**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A34x2.bp.terminal_1**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A34x2.bp.terminal_2**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A34x2.bp.terminal_3**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A34x2.bp.trusted_key_storage**
Trusted Root-Key Storage unit.
Type: `RootKeyStorage`.

**FVP_Base_Cortex_A34x2.bp.trusted_nv_counter**
Trusted Non-Volatile Counter unit.
Type: `NonVolatileCounter`.

**FVP_Base_Cortex_A34x2.bp.trusted_rng**
Random Number Generator unit.
Type: `RandomNumberGenerator`.

**FVP_Base_Cortex_A34x2.bp.trusted_watchdog**
ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A34x2.bp.trusted_watchdog.clocktimer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A34x2.bp.trusted_watchdog.clocktimer.timer**
A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A34x2.bp.trusted_watchdog.clocktimer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A34x2.bp.trusted_watchdog.clocktimer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A34x2.bp.tzc_400**
TrustZone Address Space Controller.
Type: `TZC_400`.

**FVP_Base_Cortex_A34x2.bp.ve_sysregs**
Type: `VE_SysRegs`.

**FVP_Base_Cortex_A34x2.bp.vedcc**
Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

**FVP_Base_Cortex_A34x2.bp.virtio_net**
VirtioNet device over MMIO transport.
Type: `VirtioNetMMIO`.

**FVP_Base_Cortex_A34x2.bp.virtio_net_labeller**
Type: `Labeller`.

**FVP_Base_Cortex_A34x2.bp.virtio_rng**
VirtioEntropy device over MMIO transport.
Type: `VirtioEntropyMMIO`.

**FVP_Base_Cortex_A34x2.bp.virtioblockdevice**
VirtioBlock device over MMIO transport.
Type: `VirtioBlockMMIO`.

**FVP_Base_Cortex_A34x2.bp.virtioblockdevice_labeller**
Type: `Labeller`.

**FVP_Base_Cortex_A34x2.bp.virtiop9device**
virtio P9 server.
Type: `VirtioP9Device`.

**FVP_Base_Cortex_A34x2.bp.virtiop9device_labeller**
Type: `Labeller`.

**FVP_Base_Cortex_A34x2.bp.vis**
Display window for VE using Visualisation library.
Type: `VEVisualisation`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A34x2.bp.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_Base_Cortex_A34x2.bp.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A34x2.bp.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A34x2.bp.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A34x2.bp.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A34x2.bp.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A34x2.bp.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A34x2.bp.vram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A34x2.cci400**

Cache Coherent Interconnect for AXI4 ACE.
Type: `CCI400`.

**FVP_Base_Cortex_A34x2.clockdivider0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A34x2.cluster0**

ARM Cortex-A34 Cluster CT model.
Type: `Cluster_ARM_Cortex-A34`.

**FVP_Base_Cortex_A34x2.cluster0.cpu0**

ARM Cortex-A34 CT model.
Type: `ARM_Cortex-A34`.

**FVP_Base_Cortex_A34x2.cluster0.cpu0.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A34x2.cluster0.cpu0.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A34x2.cluster0.cpu0.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A34x2.cluster0.cpu1**

ARM Cortex-A34 CT model.
Type: `ARM_Cortex-A34`.

**FVP_Base_Cortex_A34x2.cluster0.cpu1.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A34x2.cluster0.cpu1.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A34x2.cluster0.cpu1.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A34x2.cluster0.l2_cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A34x2.cluster0_labeller**

Type: `Labeller`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A34x2.dapmemlogger**

Bus Logger.
Type: `PVBusLogger`.

**FVP_Base_Cortex_A34x2.elfloader**

ELF loader component.
Type: `ElfLoader`.

**FVP_Base_Cortex_A34x2.gic_distributor**

GIC Interrupt Redistribution Infrastucture component.
Type: `GIC_IRI`.

**FVP_Base_Cortex_A34x2.pctl**

Base Platforms Power Controller.
Type: `Base_PowerController`.

**FVP_Base_Cortex_A34x2.pctl.timer_reset**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A34x2.pctl.timer_reset.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A34x2.pctl.timer_reset.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A34x2.pctl.timer_reset.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

# 13.8 FVP_Base_Cortex-A34x4

List of instances in FVP_Base_Cortex-A34x4.

## FVP_Base_Cortex-A34x4 instances

**FVP_Base_Cortex_A34x4**

Base Platform Compute Subsystem for ARMCortexA34x4CT.
Type: `FVP_Base_Cortex_A34x4`.

**FVP_Base_Cortex_A34x4.bp**

Peripherals and address map for the Base Platform.
Type: `BasePlatformPeripherals`.

**FVP_Base_Cortex_A34x4.bp.Timer_0_1**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Cortex_A34x4.bp.Timer_0_1.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A34x4.bp.Timer_0_1.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A34x4.bp.Timer_0_1.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A34x4.bp.Timer_0_1.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A34x4.bp.Timer_2_3**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Cortex_A34x4.bp.Timer_2_3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A34x4.bp.Timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A34x4.bp.Timer_2_3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A34x4.bp.Timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A34x4.bp.ap_refclk**

ARM Generic Timer.
Type: `MemoryMappedGenericTimer`.

**FVP_Base_Cortex_A34x4.bp.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**FVP_Base_Cortex_A34x4.bp.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A34x4.bp.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A34x4.bp.clock300MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A34x4.bp.clock32KHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A34x4.bp.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A34x4.bp.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A34x4.bp.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A34x4.bp.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A34x4.bp.dmc**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A34x4.bp.dmc_phy**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A34x4.bp.dummy_local_dap_rom**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A34x4.bp.dummy_ram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A34x4.bp.dummy_usb**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A34x4.bp.exclusive_monitor**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A34x4.bp.flash0**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A34x4.bp.flash1**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A34x4.bp.flashloader0**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A34x4.bp.flashloader1**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A34x4.bp.generic_watchdog**

ARM Generic Watchdog.
Type: `MemoryMappedGenericWatchdog`.

**FVP_Base_Cortex_A34x4.bp.hdlcd0**

ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).
Type: `PL370_HDLCD`.

**FVP_Base_Cortex_A34x4.bp.hdlcd0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A34x4.bp.hdlcd0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A34x4.bp.hdlcd0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A34x4.bp.hdlcd0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A34x4.bp.hdlcd0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A34x4.bp.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_Base_Cortex_A34x4.bp.mmc**

Generic Multimedia Card.
Type: `MMC`.

**FVP_Base_Cortex_A34x4.bp.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A34x4.bp.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A34x4.bp.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A34x4.bp.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A34x4.bp.mpe**

Encrypt memory transactions for each encryption context with an independent key to prevent mismatch access.
Type: `PVMemoryProtectionEngine`.

**FVP_Base_Cortex_A34x4.bp.nontrustedrom**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A34x4.bp.nontrustedromloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A34x4.bp.ns_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A34x4.bp.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A34x4.bp.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A34x4.bp.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A34x4.bp.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A34x4.bp.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A34x4.bp.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A34x4.bp.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A34x4.bp.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A34x4.bp.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A34x4.bp.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A34x4.bp.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A34x4.bp.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A34x4.bp.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A34x4.bp.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A34x4.bp.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ClockTimerThread`.

**FVP_Base_Cortex_A34x4.bp.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A34x4.bp.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A34x4.bp.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A34x4.bp.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A34x4.bp.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A34x4.bp.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A34x4.bp.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A34x4.bp.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A34x4.bp.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A34x4.bp.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

**FVP_Base_Cortex_A34x4.bp.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A34x4.bp.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A34x4.bp.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A34x4.bp.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A34x4.bp.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_Base_Cortex_A34x4.bp.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A34x4.bp.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A34x4.bp.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A34x4.bp.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A34x4.bp.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A34x4.bp.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A34x4.bp.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A34x4.bp.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A34x4.bp.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_Base_Cortex_A34x4.bp.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A34x4.bp.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A34x4.bp.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A34x4.bp.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A34x4.bp.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A34x4.bp.pl111_clcd_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A34x4.bp.pl180_mci**

ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

**FVP_Base_Cortex_A34x4.bp.ps2keyboard**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

**FVP_Base_Cortex_A34x4.bp.ps2keyboard.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A34x4.bp.ps2keyboard.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A34x4.bp.ps2keyboard.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A34x4.bp.ps2keyboard.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A34x4.bp.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

**FVP_Base_Cortex_A34x4.bp.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A34x4.bp.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A34x4.bp.ps2mouse.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A34x4.bp.ps2mouse.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A34x4.bp.psram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A34x4.bp.refcounter**
Memory Mapped Counter Module for Generic Timers.
Type: `MemoryMappedCounterModule`.

**FVP_Base_Cortex_A34x4.bp.reset_or**
Or Gate.
Type: `OrGate`.

**FVP_Base_Cortex_A34x4.bp.rl_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A34x4.bp.rt_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A34x4.bp.s_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A34x4.bp.secureDRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A34x4.bp.secureSRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A34x4.bp.secureflash**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A34x4.bp.secureflashloader**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A34x4.bp.smsc_91c111**
SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_Base_Cortex_A34x4.bp.sp805_wdog**
ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A34x4.bp.sp805_wdog.clocktimer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A34x4.bp.sp805_wdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A34x4.bp.sp805_wdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A34x4.bp.sp805_wdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A34x4.bp.sp810_sysctrl**

Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_Base_Cortex_A34x4.bp.sp810_sysctrl.clkdiv_clk0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A34x4.bp.sp810_sysctrl.clkdiv_clk1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A34x4.bp.sp810_sysctrl.clkdiv_clk2**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A34x4.bp.sp810_sysctrl.clkdiv_clk3**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A34x4.bp.sram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A34x4.bp.terminal_0**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A34x4.bp.terminal_1**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A34x4.bp.terminal_2**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A34x4.bp.terminal_3**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A34x4.bp.trusted_key_storage**
Trusted Root-Key Storage unit.
Type: `RootKeyStorage`.

**FVP_Base_Cortex_A34x4.bp.trusted_nv_counter**
Trusted Non-Volatile Counter unit.
Type: `NonVolatileCounter`.

**FVP_Base_Cortex_A34x4.bp.trusted_rng**
Random Number Generator unit.
Type: `RandomNumberGenerator`.

**FVP_Base_Cortex_A34x4.bp.trusted_watchdog**
ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A34x4.bp.trusted_watchdog.clocktimer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A34x4.bp.trusted_watchdog.clocktimer.timer**
A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A34x4.bp.trusted_watchdog.clocktimer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A34x4.bp.trusted_watchdog.clocktimer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A34x4.bp.tzc_400**
TrustZone Address Space Controller.
Type: `TZC_400`.

**FVP_Base_Cortex_A34x4.bp.ve_sysregs**
Type: `VE_SysRegs`.

**FVP_Base_Cortex_A34x4.bp.vedcc**
Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

**FVP_Base_Cortex_A34x4.bp.virtio_net**
VirtioNet device over MMIO transport.
Type: `VirtioNetMMIO`.

**FVP_Base_Cortex_A34x4.bp.virtio_net_labeller**
Type: `Labeller`.

**FVP_Base_Cortex_A34x4.bp.virtio_rng**
VirtioEntropy device over MMIO transport.
Type: `VirtioEntropyMMIO`.

**FVP_Base_Cortex_A34x4.bp.virtioblockdevice**
VirtioBlock device over MMIO transport.
Type: `VirtioBlockMMIO`.

**FVP_Base_Cortex_A34x4.bp.virtioblockdevice_labeller**
Type: `Labeller`.

**FVP_Base_Cortex_A34x4.bp.virtiop9device**
virtio P9 server.
Type: `VirtioP9Device`.

**FVP_Base_Cortex_A34x4.bp.virtiop9device_labeller**
Type: `Labeller`.

**FVP_Base_Cortex_A34x4.bp.vis**
Display window for VE using Visualisation library.
Type: `VEVisualisation`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A34x4.bp.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_Base_Cortex_A34x4.bp.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A34x4.bp.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A34x4.bp.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A34x4.bp.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A34x4.bp.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A34x4.bp.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A34x4.bp.vram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A34x4.cci400**

Cache Coherent Interconnect for AXI4 ACE.
Type: `CCI400`.

**FVP_Base_Cortex_A34x4.clockdivider0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A34x4.cluster0**

ARM Cortex-A34 Cluster CT model.
Type: `Cluster_ARM_Cortex-A34`.

**FVP_Base_Cortex_A34x4.cluster0.cpu0**

ARM Cortex-A34 CT model.
Type: `ARM_Cortex-A34`.

**FVP_Base_Cortex_A34x4.cluster0.cpu0.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A34x4.cluster0.cpu0.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A34x4.cluster0.cpu0.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A34x4.cluster0.cpu1**

ARM Cortex-A34 CT model.
Type: `ARM_Cortex-A34`.

**FVP_Base_Cortex_A34x4.cluster0.cpu1.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A34x4.cluster0.cpu1.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A34x4.cluster0.cpu1.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A34x4.cluster0.cpu2**

ARM Cortex-A34 CT model.
Type: `ARM_Cortex-A34`.

**FVP_Base_Cortex_A34x4.cluster0.cpu2.dtlb**

TLB - instruction, data or unified.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `TlbCadi`.

**FVP_Base_Cortex_A34x4.cluster0.cpu2.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A34x4.cluster0.cpu2.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A34x4.cluster0.cpu3**

ARM Cortex-A34 CT model.
Type: `ARM_Cortex-A34`.

**FVP_Base_Cortex_A34x4.cluster0.cpu3.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A34x4.cluster0.cpu3.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A34x4.cluster0.cpu3.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A34x4.cluster0.l2_cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A34x4.cluster0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A34x4.dapmemlogger**

Bus Logger.
Type: `PVBusLogger`.

**FVP_Base_Cortex_A34x4.elfloader**

ELF loader component.
Type: `ElfLoader`.

**FVP_Base_Cortex_A34x4.gic_distributor**

GIC Interrupt Redistribution Infrastucture component.
Type: `GIC_IRI`.

**FVP_Base_Cortex_A34x4.pctl**

Base Platforms Power Controller.
Type: `Base_PowerController`.

**FVP_Base_Cortex_A34x4.pctl.timer_reset**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A34x4.pctl.timer_reset.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A34x4.pctl.timer_reset.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A34x4.pctl.timer_reset.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

## 13.9 FVP_Base_Cortex-A35x1

List of instances in FVP_Base_Cortex-A35x1.

### FVP_Base_Cortex-A35x1 instances

**FVP_Base_Cortex_A35x1**

Base Platform Compute Subsystem for ARMCortexA35x1CT.
Type: `FVP_Base_Cortex_A35x1`.

**FVP_Base_Cortex_A35x1.bp**

Peripherals and address map for the Base Platform.
Type: `BasePlatformPeripherals`.

**FVP_Base_Cortex_A35x1.bp.Timer_0_1**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Cortex_A35x1.bp.Timer_0_1.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A35x1.bp.Timer_0_1.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A35x1.bp.Timer_0_1.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A35x1.bp.Timer_0_1.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A35x1.bp.Timer_2_3**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Cortex_A35x1.bp.Timer_2_3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A35x1.bp.Timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A35x1.bp.Timer_2_3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A35x1.bp.Timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A35x1.bp.ap_refclk**

ARM Generic Timer.
Type: `MemoryMappedGenericTimer`.

**FVP_Base_Cortex_A35x1.bp.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**FVP_Base_Cortex_A35x1.bp.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A35x1.bp.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A35x1.bp.clock300MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A35x1.bp.clock32KHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A35x1.bp.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A35x1.bp.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A35x1.bp.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A35x1.bp.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ClockDivider`.

**FVP_Base_Cortex_A35x1.bp.dmc**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A35x1.bp.dmc_phy**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A35x1.bp.dummy_local_dap_rom**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A35x1.bp.dummy_ram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A35x1.bp.dummy_usb**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A35x1.bp.exclusive_monitor**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_Base_Cortex_A35x1.bp.flash0**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A35x1.bp.flash1**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A35x1.bp.flashloader0**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A35x1.bp.flashloader1**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A35x1.bp.generic_watchdog**

ARM Generic Watchdog.
Type: `MemoryMappedGenericWatchdog`.

**FVP_Base_Cortex_A35x1.bp.hdlcd0**

ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).
Type: `PL370_HDLCD`.

**FVP_Base_Cortex_A35x1.bp.hdlcd0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A35x1.bp.hdlcd0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A35x1.bp.hdlcd0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A35x1.bp.hdlcd0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A35x1.bp.hdlcd0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A35x1.bp.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_Base_Cortex_A35x1.bp.mmc**

Generic Multimedia Card.
Type: `MMC`.

**FVP_Base_Cortex_A35x1.bp.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A35x1.bp.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A35x1.bp.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A35x1.bp.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A35x1.bp.mpe**

Encrypt memory transactions for each encryption context with an independent key to prevent mismatch access.
Type: `PVMemoryProtectionEngine`.

**FVP_Base_Cortex_A35x1.bp.nontrustedrom**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A35x1.bp.nontrustedromloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A35x1.bp.ns_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A35x1.bp.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A35x1.bp.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A35x1.bp.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A35x1.bp.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A35x1.bp.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A35x1.bp.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A35x1.bp.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A35x1.bp.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A35x1.bp.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A35x1.bp.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A35x1.bp.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A35x1.bp.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A35x1.bp.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A35x1.bp.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A35x1.bp.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A35x1.bp.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A35x1.bp.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A35x1.bp.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A35x1.bp.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A35x1.bp.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A35x1.bp.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A35x1.bp.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A35x1.bp.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A35x1.bp.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A35x1.bp.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

**FVP_Base_Cortex_A35x1.bp.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A35x1.bp.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A35x1.bp.pl031_rtc.timer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A35x1.bp.pl031_rtc.timer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A35x1.bp.pl041_aaci**
ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_Base_Cortex_A35x1.bp.pl041_aaci.timer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A35x1.bp.pl041_aaci.timer.timer**
A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A35x1.bp.pl041_aaci.timer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A35x1.bp.pl041_aaci.timer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A35x1.bp.pl050_kmi0**
ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A35x1.bp.pl050_kmi0.clk_divider**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A35x1.bp.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A35x1.bp.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A35x1.bp.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_Base_Cortex_A35x1.bp.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

**FVP_Base_Cortex_A35x1.bp.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A35x1.bp.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A35x1.bp.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A35x1.bp.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A35x1.bp.pl111_clcd_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A35x1.bp.pl180_mci**

ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

**FVP_Base_Cortex_A35x1.bp.ps2keyboard**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

**FVP_Base_Cortex_A35x1.bp.ps2keyboard.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A35x1.bp.ps2keyboard.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A35x1.bp.ps2keyboard.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A35x1.bp.ps2keyboard.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A35x1.bp.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

**FVP_Base_Cortex_A35x1.bp.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A35x1.bp.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A35x1.bp.ps2mouse.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A35x1.bp.ps2mouse.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A35x1.bp.psram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A35x1.bp.refcounter**

Memory Mapped Counter Module for Generic Timers.
Type: `MemoryMappedCounterModule`.

**FVP_Base_Cortex_A35x1.bp.reset_or**

Or Gate.
Type: `OrGate`.

**FVP_Base_Cortex_A35x1.bp.rl_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A35x1.bp.rt_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A35x1.bp.s_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A35x1.bp.secureDRAM**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A35x1.bp.secureSRAM**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A35x1.bp.secureflash**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A35x1.bp.secureflashloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A35x1.bp.smsc_91c111**

SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_Base_Cortex_A35x1.bp.sp805_wdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A35x1.bp.sp805_wdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A35x1.bp.sp805_wdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A35x1.bp.sp805_wdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A35x1.bp.sp805_wdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A35x1.bp.sp810_sysctrl**

Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A35x1.bp.sp810_sysctrl.clkdiv_clk0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A35x1.bp.sp810_sysctrl.clkdiv_clk1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A35x1.bp.sp810_sysctrl.clkdiv_clk2**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A35x1.bp.sp810_sysctrl.clkdiv_clk3**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A35x1.bp.sram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A35x1.bp.terminal_0**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A35x1.bp.terminal_1**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A35x1.bp.terminal_2**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A35x1.bp.terminal_3**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A35x1.bp.trusted_key_storage**

Trusted Root-Key Storage unit.
Type: `RootKeyStorage`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A35x1.bp.trusted_nv_counter**

Trusted Non-Volatile Counter unit.
Type: `NonVolatileCounter`.

**FVP_Base_Cortex_A35x1.bp.trusted_rng**

Random Number Generator unit.
Type: `RandomNumberGenerator`.

**FVP_Base_Cortex_A35x1.bp.trusted_watchdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A35x1.bp.trusted_watchdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A35x1.bp.trusted_watchdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A35x1.bp.trusted_watchdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A35x1.bp.trusted_watchdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A35x1.bp.tzc_400**

TrustZone Address Space Controller.
Type: `TZC_400`.

**FVP_Base_Cortex_A35x1.bp.ve_sysregs**

Type: `VE_SysRegs`.

**FVP_Base_Cortex_A35x1.bp.vedcc**

Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A35x1.bp.virtio_net**

VirtioNet device over MMIO transport.
Type: `VirtioNetMMIO`.

**FVP_Base_Cortex_A35x1.bp.virtio_net_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A35x1.bp.virtio_rng**

VirtioEntropy device over MMIO transport.
Type: `VirtioEntropyMMIO`.

**FVP_Base_Cortex_A35x1.bp.virtioblockdevice**

VirtioBlock device over MMIO transport.
Type: `VirtioBlockMMIO`.

**FVP_Base_Cortex_A35x1.bp.virtioblockdevice_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A35x1.bp.virtiop9device**

virtio P9 server.
Type: `VirtioP9Device`.

**FVP_Base_Cortex_A35x1.bp.virtiop9device_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A35x1.bp.vis**

Display window for VE using Visualisation library.
Type: `VEVisualisation`.

**FVP_Base_Cortex_A35x1.bp.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_Base_Cortex_A35x1.bp.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A35x1.bp.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A35x1.bp.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A35x1.bp.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A35x1.bp.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A35x1.bp.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A35x1.bp.vram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A35x1.cci400**

Cache Coherent Interconnect for AXI4 ACE.
Type: `CCI400`.

**FVP_Base_Cortex_A35x1.clockdivider0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A35x1.cluster0**

ARM Cortex-A35 Cluster CT model.
Type: `Cluster_ARM_Cortex-A35`.

**FVP_Base_Cortex_A35x1.cluster0.cpu0**

ARM Cortex-A35 CT model.
Type: `ARM_Cortex-A35`.

**FVP_Base_Cortex_A35x1.cluster0.cpu0.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A35x1.cluster0.cpu0.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A35x1.cluster0.cpu0.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A35x1.cluster0.l2_cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A35x1.cluster0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A35x1.dapmemlogger**

Bus Logger.
Type: `PVBusLogger`.

**FVP_Base_Cortex_A35x1.elfloader**

ELF loader component.
Type: `ElfLoader`.

**FVP_Base_Cortex_A35x1.gic_distributor**

GIC Interrupt Redistribution Infrastucture component.
Type: `GIC_IRI`.

**FVP_Base_Cortex_A35x1.pctl**

Base Platforms Power Controller.
Type: `Base_PowerController`.

**FVP_Base_Cortex_A35x1.pctl.timer_reset**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A35x1.pctl.timer_reset.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A35x1.pctl.timer_reset.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `SchedulerThread`.

**FVP_Base_Cortex_A35x1.pctl.timer_reset.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

# 13.10 FVP_Base_Cortex-A35x2

List of instances in FVP_Base_Cortex-A35x2.

## FVP_Base_Cortex-A35x2 instances

**FVP_Base_Cortex_A35x2**

Base Platform Compute Subsystem for ARMCortexA35x2CT.
Type: `FVP_Base_Cortex_A35x2`.

**FVP_Base_Cortex_A35x2.bp**

Peripherals and address map for the Base Platform.
Type: `BasePlatformPeripherals`.

**FVP_Base_Cortex_A35x2.bp.Timer_0_1**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Cortex_A35x2.bp.Timer_0_1.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A35x2.bp.Timer_0_1.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A35x2.bp.Timer_0_1.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A35x2.bp.Timer_0_1.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A35x2.bp.Timer_2_3**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A35x2.bp.Timer_2_3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A35x2.bp.Timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A35x2.bp.Timer_2_3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A35x2.bp.Timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A35x2.bp.ap_refclk**

ARM Generic Timer.
Type: `MemoryMappedGenericTimer`.

**FVP_Base_Cortex_A35x2.bp.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**FVP_Base_Cortex_A35x2.bp.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A35x2.bp.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A35x2.bp.clock300MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A35x2.bp.clock32KHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A35x2.bp.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A35x2.bp.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A35x2.bp.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A35x2.bp.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A35x2.bp.dmc**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A35x2.bp.dmc_phy**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A35x2.bp.dummy_local_dap_rom**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A35x2.bp.dummy_ram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A35x2.bp.dummy_usb**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A35x2.bp.exclusive_monitor**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_Base_Cortex_A35x2.bp.flash0**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A35x2.bp.flash1**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A35x2.bp.flashloader0**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A35x2.bp.flashloader1**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A35x2.bp.generic_watchdog**

ARM Generic Watchdog.
Type: `MemoryMappedGenericWatchdog`.

**FVP_Base_Cortex_A35x2.bp.hdlcd0**

ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).
Type: `PL370_HDLCD`.

**FVP_Base_Cortex_A35x2.bp.hdlcd0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A35x2.bp.hdlcd0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A35x2.bp.hdlcd0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A35x2.bp.hdlcd0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A35x2.bp.hdlcd0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A35x2.bp.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_Base_Cortex_A35x2.bp.mmc**

Generic Multimedia Card.
Type: `MMC`.

**FVP_Base_Cortex_A35x2.bp.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A35x2.bp.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A35x2.bp.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A35x2.bp.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A35x2.bp.mpe**

Encrypt memory transactions for each encryption context with an independent key to prevent mismatch access.
Type: `PVMemoryProtectionEngine`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A35x2.bp.nontrustedrom**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A35x2.bp.nontrustedromloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A35x2.bp.ns_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A35x2.bp.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A35x2.bp.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A35x2.bp.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A35x2.bp.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A35x2.bp.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A35x2.bp.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A35x2.bp.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A35x2.bp.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A35x2.bp.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A35x2.bp.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A35x2.bp.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A35x2.bp.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A35x2.bp.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A35x2.bp.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A35x2.bp.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A35x2.bp.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A35x2.bp.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A35x2.bp.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A35x2.bp.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A35x2.bp.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A35x2.bp.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A35x2.bp.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A35x2.bp.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A35x2.bp.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A35x2.bp.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

**FVP_Base_Cortex_A35x2.bp.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A35x2.bp.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A35x2.bp.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A35x2.bp.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A35x2.bp.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_Base_Cortex_A35x2.bp.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A35x2.bp.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A35x2.bp.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A35x2.bp.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A35x2.bp.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A35x2.bp.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A35x2.bp.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A35x2.bp.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A35x2.bp.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_Base_Cortex_A35x2.bp.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

**FVP_Base_Cortex_A35x2.bp.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A35x2.bp.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A35x2.bp.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A35x2.bp.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A35x2.bp.pl111_clcd_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A35x2.bp.pl180_mci**

ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

**FVP_Base_Cortex_A35x2.bp.ps2keyboard**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A35x2.bp.ps2keyboard.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A35x2.bp.ps2keyboard.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A35x2.bp.ps2keyboard.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A35x2.bp.ps2keyboard.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A35x2.bp.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

**FVP_Base_Cortex_A35x2.bp.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A35x2.bp.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A35x2.bp.ps2mouse.ps2_clocktimer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A35x2.bp.ps2mouse.ps2_clocktimer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A35x2.bp.psram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A35x2.bp.refcounter**
Memory Mapped Counter Module for Generic Timers.
Type: `MemoryMappedCounterModule`.

**FVP_Base_Cortex_A35x2.bp.reset_or**
Or Gate.
Type: `OrGate`.

**FVP_Base_Cortex_A35x2.bp.rl_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A35x2.bp.rt_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A35x2.bp.s_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A35x2.bp.secureDRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A35x2.bp.secureSRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A35x2.bp.secureflash**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A35x2.bp.secureflashloader**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A35x2.bp.smsc_91c111**

SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_Base_Cortex_A35x2.bp.sp805_wdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A35x2.bp.sp805_wdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A35x2.bp.sp805_wdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A35x2.bp.sp805_wdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A35x2.bp.sp805_wdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A35x2.bp.sp810_sysctrl**

Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_Base_Cortex_A35x2.bp.sp810_sysctrl.clkdiv_clk0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A35x2.bp.sp810_sysctrl.clkdiv_clk1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A35x2.bp.sp810_sysctrl.clkdiv_clk2**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A35x2.bp.sp810_sysctrl.clkdiv_clk3**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A35x2.bp.sram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A35x2.bp.terminal_0**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A35x2.bp.terminal_1**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A35x2.bp.terminal_2**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A35x2.bp.terminal_3**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A35x2.bp.trusted_key_storage**
Trusted Root-Key Storage unit.
Type: `RootKeyStorage`.

**FVP_Base_Cortex_A35x2.bp.trusted_nv_counter**
Trusted Non-Volatile Counter unit.
Type: `NonVolatileCounter`.

**FVP_Base_Cortex_A35x2.bp.trusted_rng**
Random Number Generator unit.
Type: `RandomNumberGenerator`.

**FVP_Base_Cortex_A35x2.bp.trusted_watchdog**
ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A35x2.bp.trusted_watchdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A35x2.bp.trusted_watchdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A35x2.bp.trusted_watchdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A35x2.bp.trusted_watchdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A35x2.bp.tzc_400**

TrustZone Address Space Controller.
Type: `TZC_400`.

**FVP_Base_Cortex_A35x2.bp.ve_sysregs**

Type: `VE_SysRegs`.

**FVP_Base_Cortex_A35x2.bp.vedcc**

Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

**FVP_Base_Cortex_A35x2.bp.virtio_net**

VirtioNet device over MMIO transport.
Type: `VirtioNetMMIO`.

**FVP_Base_Cortex_A35x2.bp.virtio_net_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A35x2.bp.virtio_rng**

VirtioEntropy device over MMIO transport.
Type: `VirtioEntropyMMIO`.

**FVP_Base_Cortex_A35x2.bp.virtioblockdevice**

VirtioBlock device over MMIO transport.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `VirtioBlockMMIO`.

**FVP_Base_Cortex_A35x2.bp.virtioblockdevice_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A35x2.bp.virtiop9device**

virtio P9 server.
Type: `VirtioP9Device`.

**FVP_Base_Cortex_A35x2.bp.virtiop9device_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A35x2.bp.vis**

Display window for VE using Visualisation library.
Type: `VEVisualisation`.

**FVP_Base_Cortex_A35x2.bp.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_Base_Cortex_A35x2.bp.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A35x2.bp.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A35x2.bp.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A35x2.bp.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A35x2.bp.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A35x2.bp.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A35x2.bp.vram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A35x2.cci400**

Cache Coherent Interconnect for AXI4 ACE.
Type: `CCI400`.

**FVP_Base_Cortex_A35x2.clockdivider0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A35x2.cluster0**

ARM Cortex-A35 Cluster CT model.
Type: `Cluster_ARM_Cortex-A35`.

**FVP_Base_Cortex_A35x2.cluster0.cpu0**

ARM Cortex-A35 CT model.
Type: `ARM_Cortex-A35`.

**FVP_Base_Cortex_A35x2.cluster0.cpu0.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A35x2.cluster0.cpu0.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A35x2.cluster0.cpu0.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A35x2.cluster0.cpu1**

ARM Cortex-A35 CT model.
Type: `ARM_Cortex-A35`.

**FVP_Base_Cortex_A35x2.cluster0.cpu1.dtlb**

TLB - instruction, data or unified.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `TlbCadi`.

**FVP_Base_Cortex_A35x2.cluster0.cpu1.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A35x2.cluster0.cpu1.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A35x2.cluster0.l2_cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A35x2.cluster0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A35x2.dapmemlogger**

Bus Logger.
Type: `PVBusLogger`.

**FVP_Base_Cortex_A35x2.elfloader**

ELF loader component.
Type: `ElfLoader`.

**FVP_Base_Cortex_A35x2.gic_distributor**

GIC Interrupt Redistribution Infrastucture component.
Type: `GIC_IRI`.

**FVP_Base_Cortex_A35x2.pctl**

Base Platforms Power Controller.
Type: `Base_PowerController`.

**FVP_Base_Cortex_A35x2.pctl.timer_reset**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A35x2.pctl.timer_reset.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A35x2.pctl.timer_reset.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A35x2.pctl.timer_reset.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

## 13.11 FVP_Base_Cortex-A35x4

List of instances in FVP_Base_Cortex-A35x4.

### FVP_Base_Cortex-A35x4 instances

**FVP_Base_Cortex_A35x4**

Base Platform Compute Subsystem for ARMCortexA35x4CT.
Type: `FVP_Base_Cortex_A35x4`.

**FVP_Base_Cortex_A35x4.bp**

Peripherals and address map for the Base Platform.
Type: `BasePlatformPeripherals`.

**FVP_Base_Cortex_A35x4.bp.Timer_0_1**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Cortex_A35x4.bp.Timer_0_1.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A35x4.bp.Timer_0_1.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A35x4.bp.Timer_0_1.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A35x4.bp.Timer_0_1.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A35x4.bp.Timer_2_3**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Cortex_A35x4.bp.Timer_2_3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A35x4.bp.Timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A35x4.bp.Timer_2_3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A35x4.bp.Timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A35x4.bp.ap_refclk**

ARM Generic Timer.
Type: `MemoryMappedGenericTimer`.

**FVP_Base_Cortex_A35x4.bp.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**FVP_Base_Cortex_A35x4.bp.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A35x4.bp.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A35x4.bp.clock300MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A35x4.bp.clock32KHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A35x4.bp.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A35x4.bp.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A35x4.bp.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A35x4.bp.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A35x4.bp.dmc**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A35x4.bp.dmc_phy**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A35x4.bp.dummy_local_dap_rom**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A35x4.bp.dummy_ram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A35x4.bp.dummy_usb**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A35x4.bp.exclusive_monitor**
Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_Base_Cortex_A35x4.bp.flash0**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A35x4.bp.flash1**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A35x4.bp.flashloader0**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A35x4.bp.flashloader1**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A35x4.bp.generic_watchdog**
ARM Generic Watchdog.
Type: `MemoryMappedGenericWatchdog`.

**FVP_Base_Cortex_A35x4.bp.hdlcd0**
ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).
Type: `PL370_HDLCD`.

**FVP_Base_Cortex_A35x4.bp.hdlcd0.timer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A35x4.bp.hdlcd0.timer.timer**
A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A35x4.bp.hdlcd0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A35x4.bp.hdlcd0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A35x4.bp.hdlcd0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A35x4.bp.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_Base_Cortex_A35x4.bp.mmc**

Generic Multimedia Card.
Type: `MMC`.

**FVP_Base_Cortex_A35x4.bp.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A35x4.bp.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A35x4.bp.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A35x4.bp.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A35x4.bp.mpe**

Encrypt memory transactions for each encryption context with an independent key to prevent mismatch access.
Type: `PVMemoryProtectionEngine`.

**FVP_Base_Cortex_A35x4.bp.nontrustedrom**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A35x4.bp.nontrustedromloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A35x4.bp.ns_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A35x4.bp.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A35x4.bp.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A35x4.bp.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A35x4.bp.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A35x4.bp.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A35x4.bp.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A35x4.bp.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A35x4.bp.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A35x4.bp.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A35x4.bp.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A35x4.bp.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A35x4.bp.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A35x4.bp.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A35x4.bp.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A35x4.bp.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A35x4.bp.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A35x4.bp.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A35x4.bp.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A35x4.bp.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A35x4.bp.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A35x4.bp.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ClockTimerThread`.

**FVP_Base_Cortex_A35x4.bp.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A35x4.bp.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A35x4.bp.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A35x4.bp.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

**FVP_Base_Cortex_A35x4.bp.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A35x4.bp.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A35x4.bp.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A35x4.bp.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A35x4.bp.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_Base_Cortex_A35x4.bp.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A35x4.bp.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A35x4.bp.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A35x4.bp.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A35x4.bp.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A35x4.bp.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A35x4.bp.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A35x4.bp.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A35x4.bp.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_Base_Cortex_A35x4.bp.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

**FVP_Base_Cortex_A35x4.bp.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A35x4.bp.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A35x4.bp.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A35x4.bp.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A35x4.bp.pl111_clcd_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A35x4.bp.pl180_mci**

ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

**FVP_Base_Cortex_A35x4.bp.ps2keyboard**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A35x4.bp.ps2keyboard.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A35x4.bp.ps2keyboard.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A35x4.bp.ps2keyboard.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A35x4.bp.ps2keyboard.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A35x4.bp.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

**FVP_Base_Cortex_A35x4.bp.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A35x4.bp.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A35x4.bp.ps2mouse.ps2_clocktimer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A35x4.bp.ps2mouse.ps2_clocktimer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A35x4.bp.psram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A35x4.bp.refcounter**
Memory Mapped Counter Module for Generic Timers.
Type: `MemoryMappedCounterModule`.

**FVP_Base_Cortex_A35x4.bp.reset_or**
Or Gate.
Type: `OrGate`.

**FVP_Base_Cortex_A35x4.bp.rl_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A35x4.bp.rt_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A35x4.bp.s_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A35x4.bp.secureDRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A35x4.bp.secureSRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A35x4.bp.secureflash**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A35x4.bp.secureflashloader**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A35x4.bp.smsc_91c111**

SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_Base_Cortex_A35x4.bp.sp805_wdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A35x4.bp.sp805_wdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A35x4.bp.sp805_wdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A35x4.bp.sp805_wdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A35x4.bp.sp805_wdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A35x4.bp.sp810_sysctrl**

Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_Base_Cortex_A35x4.bp.sp810_sysctrl.clkdiv_clk0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A35x4.bp.sp810_sysctrl.clkdiv_clk1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A35x4.bp.sp810_sysctrl.clkdiv_clk2**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A35x4.bp.sp810_sysctrl.clkdiv_clk3**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A35x4.bp.sram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A35x4.bp.terminal_0**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A35x4.bp.terminal_1**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A35x4.bp.terminal_2**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A35x4.bp.terminal_3**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A35x4.bp.trusted_key_storage**
Trusted Root-Key Storage unit.
Type: `RootKeyStorage`.

**FVP_Base_Cortex_A35x4.bp.trusted_nv_counter**
Trusted Non-Volatile Counter unit.
Type: `NonVolatileCounter`.

**FVP_Base_Cortex_A35x4.bp.trusted_rng**
Random Number Generator unit.
Type: `RandomNumberGenerator`.

**FVP_Base_Cortex_A35x4.bp.trusted_watchdog**
ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A35x4.bp.trusted_watchdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A35x4.bp.trusted_watchdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A35x4.bp.trusted_watchdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A35x4.bp.trusted_watchdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A35x4.bp.tzc_400**

TrustZone Address Space Controller.
Type: `TZC_400`.

**FVP_Base_Cortex_A35x4.bp.ve_sysregs**

Type: `VE_SysRegs`.

**FVP_Base_Cortex_A35x4.bp.vedcc**

Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

**FVP_Base_Cortex_A35x4.bp.virtio_net**

VirtioNet device over MMIO transport.
Type: `VirtioNetMMIO`.

**FVP_Base_Cortex_A35x4.bp.virtio_net_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A35x4.bp.virtio_rng**

VirtioEntropy device over MMIO transport.
Type: `VirtioEntropyMMIO`.

**FVP_Base_Cortex_A35x4.bp.virtioblockdevice**

VirtioBlock device over MMIO transport.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `VirtioBlockMMIO`.

**FVP_Base_Cortex_A35x4.bp.virtioblockdevice_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A35x4.bp.virtiop9device**

virtio P9 server.
Type: `VirtioP9Device`.

**FVP_Base_Cortex_A35x4.bp.virtiop9device_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A35x4.bp.vis**

Display window for VE using Visualisation library.
Type: `VEVisualisation`.

**FVP_Base_Cortex_A35x4.bp.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_Base_Cortex_A35x4.bp.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A35x4.bp.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A35x4.bp.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A35x4.bp.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A35x4.bp.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A35x4.bp.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A35x4.bp.vram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A35x4.cci400**

Cache Coherent Interconnect for AXI4 ACE.
Type: `CCI400`.

**FVP_Base_Cortex_A35x4.clockdivider0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A35x4.cluster0**

ARM Cortex-A35 Cluster CT model.
Type: `Cluster_ARM_Cortex-A35`.

**FVP_Base_Cortex_A35x4.cluster0.cpu0**

ARM Cortex-A35 CT model.
Type: `ARM_Cortex-A35`.

**FVP_Base_Cortex_A35x4.cluster0.cpu0.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A35x4.cluster0.cpu0.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A35x4.cluster0.cpu0.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A35x4.cluster0.cpu1**

ARM Cortex-A35 CT model.
Type: `ARM_Cortex-A35`.

**FVP_Base_Cortex_A35x4.cluster0.cpu1.dtlb**

TLB - instruction, data or unified.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `TlbCadi`.

**FVP_Base_Cortex_A35x4.cluster0.cpu1.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A35x4.cluster0.cpu1.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A35x4.cluster0.cpu2**

ARM Cortex-A35 CT model.
Type: `ARM_Cortex-A35`.

**FVP_Base_Cortex_A35x4.cluster0.cpu2.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A35x4.cluster0.cpu2.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A35x4.cluster0.cpu2.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A35x4.cluster0.cpu3**

ARM Cortex-A35 CT model.
Type: `ARM_Cortex-A35`.

**FVP_Base_Cortex_A35x4.cluster0.cpu3.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A35x4.cluster0.cpu3.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A35x4.cluster0.cpu3.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A35x4.cluster0.l2_cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A35x4.cluster0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A35x4.dapmemlogger**

Bus Logger.
Type: `PVBusLogger`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A35x4.elfloader**

ELF loader component.
Type: `ElfLoader`.

**FVP_Base_Cortex_A35x4.gic_distributor**

GIC Interrupt Redistribution Infrastucture component.
Type: `GIC_IRI`.

**FVP_Base_Cortex_A35x4.pctl**

Base Platforms Power Controller.
Type: `Base_PowerController`.

**FVP_Base_Cortex_A35x4.pctl.timer_reset**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A35x4.pctl.timer_reset.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A35x4.pctl.timer_reset.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A35x4.pctl.timer_reset.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

## 13.12 FVP_Base_Cortex-A510

List of instances in FVP_Base_Cortex-A510.

### FVP_Base_Cortex-A510 instances

**FVP_Base_Cortex_A510**

Base Platform Compute Subsystem for ARMCortexA510CT.
Type: `FVP_Base_Cortex_A510`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A510.bp**

Peripherals and address map for the Base Platform.
Type: `BasePlatformPeripherals`.

**FVP_Base_Cortex_A510.bp.Timer_0_1**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Cortex_A510.bp.Timer_0_1.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A510.bp.Timer_0_1.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A510.bp.Timer_0_1.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A510.bp.Timer_0_1.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A510.bp.Timer_2_3**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Cortex_A510.bp.Timer_2_3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A510.bp.Timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A510.bp.Timer_2_3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A510.bp.Timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A510.bp.ap_refclk**

ARM Generic Timer.
Type: `MemoryMappedGenericTimer`.

**FVP_Base_Cortex_A510.bp.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**FVP_Base_Cortex_A510.bp.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A510.bp.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A510.bp.clock300MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A510.bp.clock32KHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A510.bp.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A510.bp.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A510.bp.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A510.bp.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A510.bp.dmc**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A510.bp.dmc_phy**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A510.bp.dummy_local_dap_rom**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A510.bp.dummy_ram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A510.bp.dummy_usb**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A510.bp.exclusive_monitor**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_Base_Cortex_A510.bp.flash0**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A510.bp.flash1**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A510.bp.flashloader0**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A510.bp.flashloader1**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A510.bp.generic_watchdog**

ARM Generic Watchdog.
Type: `MemoryMappedGenericWatchdog`.

**FVP_Base_Cortex_A510.bp.hdlcd0**

ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).
Type: `PL370_HDLCD`.

**FVP_Base_Cortex_A510.bp.hdlcd0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A510.bp.hdlcd0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A510.bp.hdlcd0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A510.bp.hdlcd0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A510.bp.hdlcd0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A510.bp.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_Base_Cortex_A510.bp.mmc**

Generic Multimedia Card.
Type: `MMC`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A510.bp.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A510.bp.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A510.bp.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A510.bp.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A510.bp.mpe**

Encrypt memory transactions for each encryption context with an independent key to prevent mismatch access.
Type: `PVMemoryProtectionEngine`.

**FVP_Base_Cortex_A510.bp.nontrustedrom**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A510.bp.nontrustedromloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A510.bp.ns_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A510.bp.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A510.bp.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A510.bp.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A510.bp.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A510.bp.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A510.bp.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A510.bp.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A510.bp.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A510.bp.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ClockTimerThread`.

**FVP_Base_Cortex_A510.bp.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A510.bp.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A510.bp.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A510.bp.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A510.bp.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A510.bp.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A510.bp.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A510.bp.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A510.bp.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A510.bp.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A510.bp.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A510.bp.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A510.bp.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A510.bp.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A510.bp.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A510.bp.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A510.bp.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A510.bp.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A510.bp.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A510.bp.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A510.bp.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_Base_Cortex_A510.bp.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A510.bp.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A510.bp.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A510.bp.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A510.bp.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A510.bp.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A510.bp.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A510.bp.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A510.bp.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_Base_Cortex_A510.bp.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

**FVP_Base_Cortex_A510.bp.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A510.bp.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A510.bp.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A510.bp.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A510.bp.pl111_clcd_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A510.bp.pl180_mci**

ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

**FVP_Base_Cortex_A510.bp.ps2keyboard**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

**FVP_Base_Cortex_A510.bp.ps2keyboard.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A510.bp.ps2keyboard.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A510.bp.ps2keyboard.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A510.bp.ps2keyboard.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A510.bp.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

**FVP_Base_Cortex_A510.bp.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A510.bp.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A510.bp.ps2mouse.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A510.bp.ps2mouse.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A510.bp.psram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A510.bp.refcounter**

Memory Mapped Counter Module for Generic Timers.
Type: `MemoryMappedCounterModule`.

**FVP_Base_Cortex_A510.bp.reset_or**

Or Gate.
Type: `OrGate`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A510.bp.rl_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A510.bp.rt_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A510.bp.s_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A510.bp.secureDRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A510.bp.secureSRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A510.bp.secureflash**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A510.bp.secureflashloader**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A510.bp.smsc_91c111**
SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_Base_Cortex_A510.bp.sp805_wdog**
ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A510.bp.sp805_wdog.clocktimer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A510.bp.sp805_wdog.clocktimer.timer**
A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A510.bp.sp805_wdog.clocktimer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A510.bp.sp805_wdog.clocktimer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A510.bp.sp810_sysctrl**
Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_Base_Cortex_A510.bp.sp810_sysctrl.clkdiv_clk0**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A510.bp.sp810_sysctrl.clkdiv_clk1**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A510.bp.sp810_sysctrl.clkdiv_clk2**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A510.bp.sp810_sysctrl.clkdiv_clk3**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A510.bp.sram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A510.bp.terminal_0**
Telnet terminal interface.
Type: `TelnetTerminal`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A510.bp.terminal_1**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A510.bp.terminal_2**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A510.bp.terminal_3**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A510.bp.trusted_key_storage**

Trusted Root-Key Storage unit.
Type: `RootKeyStorage`.

**FVP_Base_Cortex_A510.bp.trusted_nv_counter**

Trusted Non-Volatile Counter unit.
Type: `NonVolatileCounter`.

**FVP_Base_Cortex_A510.bp.trusted_rng**

Random Number Generator unit.
Type: `RandomNumberGenerator`.

**FVP_Base_Cortex_A510.bp.trusted_watchdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A510.bp.trusted_watchdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A510.bp.trusted_watchdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A510.bp.trusted_watchdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A510.bp.trusted_watchdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A510.bp.tzc_400**

TrustZone Address Space Controller.
Type: `TZC_400`.

**FVP_Base_Cortex_A510.bp.ve_sysregs**

Type: `VE_SysRegs`.

**FVP_Base_Cortex_A510.bp.vedcc**

Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

**FVP_Base_Cortex_A510.bp.virtio_net**

VirtioNet device over MMIO transport.
Type: `VirtioNetMMIO`.

**FVP_Base_Cortex_A510.bp.virtio_net_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A510.bp.virtio_rng**

VirtioEntropy device over MMIO transport.
Type: `VirtioEntropyMMIO`.

**FVP_Base_Cortex_A510.bp.virtioblockdevice**

VirtioBlock device over MMIO transport.
Type: `VirtioBlockMMIO`.

**FVP_Base_Cortex_A510.bp.virtioblockdevice_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A510.bp.virtiop9device**

virtio P9 server.
Type: `VirtioP9Device`.

**FVP_Base_Cortex_A510.bp.virtiop9device_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A510.bp.vis**

Display window for VE using Visualisation library.
Type: `VEVisualisation`.

**FVP_Base_Cortex_A510.bp.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_Base_Cortex_A510.bp.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A510.bp.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A510.bp.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A510.bp.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A510.bp.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A510.bp.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A510.bp.vram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A510.cci400**

Cache Coherent Interconnect for AXI4 ACE.
Type: `CCI400`.

**FVP_Base_Cortex_A510.clockdivider0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**`FVP_Base_Cortex_A510.cluster0`**
ARM Cortex-A510Cluster CT model.
Type: `Cluster_ARM_Cortex-A510`.

**`FVP_Base_Cortex_A510.cluster0.DSU.shared_cache`**
PV Cache.
Type: `PVCache`.

**`FVP_Base_Cortex_A510.cluster0.cpu0`**
ARM Cortex-A510CT model.
Type: `ARM_Cortex-A510`.

**`FVP_Base_Cortex_A510.cluster0.cpu0.dtlb`**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**`FVP_Base_Cortex_A510.cluster0.cpu0.l1dcache`**
PV Cache.
Type: `PVCache`.

**`FVP_Base_Cortex_A510.cluster0.cpu0.l1icache`**
PV Cache.
Type: `PVCache`.

**`FVP_Base_Cortex_A510.cluster0.cpu0.l2cache`**
PV Cache.
Type: `PVCache`.

**`FVP_Base_Cortex_A510.cluster0.cpu1`**
ARM Cortex-A510CT model.
Type: `ARM_Cortex-A510`.

**`FVP_Base_Cortex_A510.cluster0.cpu1.dtlb`**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**`FVP_Base_Cortex_A510.cluster0.cpu1.l1dcache`**
PV Cache.
Type: `PVCache`.

**`FVP_Base_Cortex_A510.cluster0.cpu1.l1icache`**
PV Cache.
Type: `PVCache`.

**`FVP_Base_Cortex_A510.cluster0.cpu1.l2cache`**
PV Cache.
Type: `PVCache`.

**`FVP_Base_Cortex_A510.cluster0.cpu10`**
ARM Cortex-A510CT model.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ARM_Cortex-A510`.

**FVP_Base_Cortex_A510.cluster0.cpu10.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A510.cluster0.cpu10.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A510.cluster0.cpu10.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A510.cluster0.cpu10.l2cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A510.cluster0.cpu11**

ARM Cortex-A510CT model.
Type: `ARM_Cortex-A510`.

**FVP_Base_Cortex_A510.cluster0.cpu11.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A510.cluster0.cpu11.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A510.cluster0.cpu11.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A510.cluster0.cpu11.l2cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A510.cluster0.cpu2**

ARM Cortex-A510CT model.
Type: `ARM_Cortex-A510`.

**FVP_Base_Cortex_A510.cluster0.cpu2.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A510.cluster0.cpu2.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A510.cluster0.cpu2.l1icache**

PV Cache.
Type: `PVCache`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A510.cluster0.cpu2.l2cache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A510.cluster0.cpu3**
ARM Cortex-A510CT model.
Type: `ARM_Cortex-A510`.

**FVP_Base_Cortex_A510.cluster0.cpu3.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A510.cluster0.cpu3.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A510.cluster0.cpu3.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A510.cluster0.cpu3.l2cache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A510.cluster0.cpu4**
ARM Cortex-A510CT model.
Type: `ARM_Cortex-A510`.

**FVP_Base_Cortex_A510.cluster0.cpu4.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A510.cluster0.cpu4.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A510.cluster0.cpu4.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A510.cluster0.cpu4.l2cache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A510.cluster0.cpu5**
ARM Cortex-A510CT model.
Type: `ARM_Cortex-A510`.

**FVP_Base_Cortex_A510.cluster0.cpu5.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A510.cluster0.cpu5.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A510.cluster0.cpu5.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A510.cluster0.cpu5.l2cache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A510.cluster0.cpu6**
ARM Cortex-A510CT model.
Type: `ARM_Cortex-A510`.

**FVP_Base_Cortex_A510.cluster0.cpu6.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A510.cluster0.cpu6.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A510.cluster0.cpu6.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A510.cluster0.cpu6.l2cache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A510.cluster0.cpu7**
ARM Cortex-A510CT model.
Type: `ARM_Cortex-A510`.

**FVP_Base_Cortex_A510.cluster0.cpu7.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A510.cluster0.cpu7.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A510.cluster0.cpu7.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A510.cluster0.cpu7.l2cache**
PV Cache.
Type: `PVCache`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A510.cluster0.cpu8**

ARM Cortex-A510CT model.
Type: ARM_Cortex-A510.

**FVP_Base_Cortex_A510.cluster0.cpu8.dtlb**

TLB - instruction, data or unified.
Type: TlbCadi.

**FVP_Base_Cortex_A510.cluster0.cpu8.l1dcache**

PV Cache.
Type: PVCache.

**FVP_Base_Cortex_A510.cluster0.cpu8.l1icache**

PV Cache.
Type: PVCache.

**FVP_Base_Cortex_A510.cluster0.cpu8.l2cache**

PV Cache.
Type: PVCache.

**FVP_Base_Cortex_A510.cluster0.cpu9**

ARM Cortex-A510CT model.
Type: ARM_Cortex-A510.

**FVP_Base_Cortex_A510.cluster0.cpu9.dtlb**

TLB - instruction, data or unified.
Type: TlbCadi.

**FVP_Base_Cortex_A510.cluster0.cpu9.l1dcache**

PV Cache.
Type: PVCache.

**FVP_Base_Cortex_A510.cluster0.cpu9.l1icache**

PV Cache.
Type: PVCache.

**FVP_Base_Cortex_A510.cluster0.cpu9.l2cache**

PV Cache.
Type: PVCache.

**FVP_Base_Cortex_A510.cluster0_labeller**

Type: Labeller.

**FVP_Base_Cortex_A510.dapmemlogger**

Bus Logger.
Type: PVBusLogger.

**FVP_Base_Cortex_A510.elfloader**

ELF loader component.
Type: ElfLoader.

**FVP_Base_Cortex_A510.gic_distributor**

GIC Interrupt Redistribution Infrastucture component.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `GIC_IRI`.

**`FVP_Base_Cortex_A510.pctl`**

Base Platforms Power Controller.
Type: `Base_PowerController`.

**`FVP_Base_Cortex_A510.pctl.timer_reset`**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**`FVP_Base_Cortex_A510.pctl.timer_reset.timer`**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**`FVP_Base_Cortex_A510.pctl.timer_reset.timer.thread`**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**`FVP_Base_Cortex_A510.pctl.timer_reset.timer.thread_event`**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

## 13.13 FVP_Base_Cortex-A510+Cortex-A710

List of instances in FVP_Base_Cortex-A510+Cortex-A710.

### FVP_Base_Cortex-A510+Cortex-A710 instances

**`FVP_Base_Cortex_A510_Cortex_A710`**

Base Platform Compute Subsystem for ARMCortexA510CT_CortexA710CT.
Type: `FVP_Base_Cortex_A510_Cortex_A710`.

**`FVP_Base_Cortex_A510_Cortex_A710.bp`**

Peripherals and address map for the Base Platform.
Type: `BasePlatformPeripherals`.

**`FVP_Base_Cortex_A510_Cortex_A710.bp.Timer_0_1`**

ARM Dual-Timer Module(SP804).

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `SP804_Timer`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.Timer_0_1.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.Timer_0_1.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.Timer_0_1.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.Timer_0_1.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.Timer_2_3**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.Timer_2_3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.Timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.Timer_2_3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.Timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.ap_refclk**

ARM Generic Timer.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `MemoryMappedGenericTimer`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.clock300MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.clock32KHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.clockdivider**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.dmc**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.dmc_phy**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.dummy_local_dap_rom**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.dummy_ram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.dummy_usb**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.exclusive_monitor**
Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.flash0**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.flash1**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.flashloader0**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.flashloader1**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.generic_watchdog**
ARM Generic Watchdog.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `MemoryMappedGenericWatchdog`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.hdlcd0**

ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).
Type: `PL370_HDLCD`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.hdlcd0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.hdlcd0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.hdlcd0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.hdlcd0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.hdlcd0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.mmc**

Generic Multimedia Card.
Type: `MMC`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.mpe**

Encrypt memory transactions for each encryption context with an independent key to prevent mismatch access.
Type: `PVMemoryProtectionEngine`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.nontrustedrom**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.nontrustedromloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.ns_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A510_Cortex_A710.bp.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A510_Cortex_A710.bp.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A510_Cortex_A710.bp.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal()

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A510_Cortex_A710.bp.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.pl111_clcd.pl11x_clcd.timer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.pl111_clcd.pl11x_clcd.timer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.pl111_clcd_labeller**
Type: `Labeller`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.pl180_mci**
ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.ps2keyboard**
Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.ps2keyboard.ps2_clocktimer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.ps2keyboard.ps2_clocktimer.timer**
A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.ps2keyboard.ps2_clocktimer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.ps2keyboard.ps2_clocktimer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.ps2mouse.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.ps2mouse.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.psram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.refcounter**

Memory Mapped Counter Module for Generic Timers.
Type: `MemoryMappedCounterModule`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.reset_or**

Or Gate.
Type: `OrGate`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.rl_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A510_Cortex_A710.bp.rt_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.s_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.secureDRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.secureSRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.secureflash**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.secureflashloader**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.smsc_91c111**
SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.sp805_wdog**
ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.sp805_wdog.clocktimer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.sp805_wdog.clocktimer.timer**
A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A510_Cortex_A710.bp.sp805_wdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.sp805_wdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.sp810_sysctrl**

Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.sp810_sysctrl.clkdiv_clk0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.sp810_sysctrl.clkdiv_clk1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.sp810_sysctrl.clkdiv_clk2**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.sp810_sysctrl.clkdiv_clk3**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.sram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.terminal_0**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.terminal_1**

Telnet terminal interface.
Type: `TelnetTerminal`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A510_Cortex_A710.bp.terminal_2**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.terminal_3**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.trusted_key_storage**

Trusted Root-Key Storage unit.
Type: `RootKeyStorage`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.trusted_nv_counter**

Trusted Non-Volatile Counter unit.
Type: `NonVolatileCounter`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.trusted_rng**

Random Number Generator unit.
Type: `RandomNumberGenerator`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.trusted_watchdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.trusted_watchdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.trusted_watchdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.trusted_watchdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.trusted_watchdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A510_Cortex_A710.bp.tzc_400**

TrustZone Address Space Controller.
Type: `TZC_400`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.ve_sysregs**

Type: `VE_SysRegs`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.vedcc**

Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.virtio_net**

VirtioNet device over MMIO transport.
Type: `VirtioNetMMIO`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.virtio_net_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.virtio_rng**

VirtioEntropy device over MMIO transport.
Type: `VirtioEntropyMMIO`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.virtioblockdevice**

VirtioBlock device over MMIO transport.
Type: `VirtioBlockMMIO`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.virtioblockdevice_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.virtiop9device**

virtio P9 server.
Type: `VirtioP9Device`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.virtiop9device_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.vis**

Display window for VE using Visualisation library.
Type: `VEVisualisation`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A510_Cortex_A710.bp.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A510_Cortex_A710.bp.vram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A510_Cortex_A710.cci400**

Cache Coherent Interconnect for AXI4 ACE.
Type: `CCI400`.

**FVP_Base_Cortex_A510_Cortex_A710.clockdivider0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A510_Cortex_A710.cluster0**

ARM Cortex-A710 CT heterogeneous cluster model.
Type: `Cluster_ARM_CortexA710_Heterogeneous_Cluster`.

**FVP_Base_Cortex_A510_Cortex_A710.cluster0.DSU.shared_cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A510_Cortex_A710.cluster0.subcluster0**

ARM Cortex-A510 CT Model in Heterogeneous Cluster.
Type: `Subcluster_ARM_Cortex-A510`.

**FVP_Base_Cortex_A510_Cortex_A710.cluster0.subcluster0.cpu0**

ARM Cortex-A510CT model.
Type: `ARM_Cortex-A510`.

**FVP_Base_Cortex_A510_Cortex_A710.cluster0.subcluster0.cpu0.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A510_Cortex_A710.cluster0.subcluster0.cpu0.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A510_Cortex_A710.cluster0.subcluster0.cpu0.l2cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A510_Cortex_A710.cluster0.subcluster1**

ARM Cortex-A710 CT Model in Heterogeneous Cluster.
Type: `Subcluster_ARM_CortexA710`.

**FVP_Base_Cortex_A510_Cortex_A710.cluster0.subcluster1.cpu0**

ARM Cortex-A710 CT model.
Type: `ARM_Cortex-A710`.

**FVP_Base_Cortex_A510_Cortex_A710.cluster0.subcluster1.cpu0.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A510_Cortex_A710.cluster0.subcluster1.cpu0.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A510_Cortex_A710.cluster0.subcluster1.cpu0.l2cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A510_Cortex_A710.cluster0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A510_Cortex_A710.dapmemlogger**

Bus Logger.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `PVBusLogger`.

**`FVP_Base_Cortex_A510_Cortex_A710.elfloader`**

ELF loader component.
Type: `ElfLoader`.

**`FVP_Base_Cortex_A510_Cortex_A710.gic_distributor`**

GIC Interrupt Redistribution Infrastucture component.
Type: `GIC_IRI`.

**`FVP_Base_Cortex_A510_Cortex_A710.pctl`**

Base Platforms Power Controller.
Type: `Base_PowerController`.

**`FVP_Base_Cortex_A510_Cortex_A710.pctl.timer_reset`**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**`FVP_Base_Cortex_A510_Cortex_A710.pctl.timer_reset.timer`**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**`FVP_Base_Cortex_A510_Cortex_A710.pctl.timer_reset.timer.thread`**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**`FVP_Base_Cortex_A510_Cortex_A710.pctl.timer_reset.timer.thread_event`**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

## 13.14 FVP_Base_Cortex-A510x4+Cortex-A710x4

List of instances in FVP_Base_Cortex-A510x4+Cortex-A710x4.

### FVP_Base_Cortex-A510x4+Cortex-A710x4 instances

**`FVP_Base_Cortex_A510x4_Cortex_A710x4`**

Base Platform Compute Subsystem for ARMCortexA510x4CT_CortexA710x4CT.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `FVP_Base_Cortex_A510x4_Cortex_A710x4`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp**

Peripherals and address map for the Base Platform.
Type: `BasePlatformPeripherals`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.Timer_0_1**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.Timer_0_1.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.Timer_0_1.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.Timer_0_1.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.Timer_0_1.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.Timer_2_3**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.Timer_2_3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.Timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.Timer_2_3.counter0**

Internal component used by SP804 Timer module.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `CounterModule`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.Timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.ap_refclk**

ARM Generic Timer.
Type: `MemoryMappedGenericTimer`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.clock300MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.clock32KHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.dmc**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.dmc_phy**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.dummy_local_dap_rom**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.dummy_ram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.dummy_usb**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.exclusive_monitor**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.flash0**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.flash1**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.flashloader0**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.flashloader1**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.generic_watchdog**

ARM Generic Watchdog.
Type: `MemoryMappedGenericWatchdog`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.hdlcd0**

ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).
Type: `PL370_HDLCD`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.hdlcd0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.hdlcd0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.hdlcd0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.hdlcd0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.hdlcd0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.mmc**

Generic Multimedia Card.
Type: `MMC`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.mpe**

Encrypt memory transactions for each encryption context with an independent key to prevent mismatch access.
Type: `PVMemoryProtectionEngine`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.nontrustedrom**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.nontrustedromloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.ns_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ClockTimerThread`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.pl111_clcd_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.pl180_mci**

ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.ps2keyboard**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.ps2keyboard.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.ps2keyboard.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.ps2keyboard.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.ps2keyboard.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.ps2mouse.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.ps2mouse.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.psram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.refcounter**

Memory Mapped Counter Module for Generic Timers.
Type: `MemoryMappedCounterModule`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.reset_or**

Or Gate.
Type: `OrGate`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.rl_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.rt_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.s_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.secureDRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.secureSRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.secureflash**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.secureflashloader**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.smsc_91c111**
SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.sp805_wdog**
ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.sp805_wdog.clocktimer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.sp805_wdog.clocktimer.timer**
A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.sp805_wdog.clocktimer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.sp805_wdog.clocktimer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.sp810_sysctrl**
Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.sp810_sysctrl.clkdiv_clk0**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.sp810_sysctrl.clkdiv_clk1**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.sp810_sysctrl.clkdiv_clk2**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.sp810_sysctrl.clkdiv_clk3**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.sram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.terminal_0**
Telnet terminal interface.
Type: `TelnetTerminal`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.terminal_1**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.terminal_2**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.terminal_3**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.trusted_key_storage**

Trusted Root-Key Storage unit.
Type: `RootKeyStorage`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.trusted_nv_counter**

Trusted Non-Volatile Counter unit.
Type: `NonVolatileCounter`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.trusted_rng**

Random Number Generator unit.
Type: `RandomNumberGenerator`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.trusted_watchdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.trusted_watchdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.trusted_watchdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.trusted_watchdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.trusted_watchdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.tzc_400**

TrustZone Address Space Controller.
Type: `TZC_400`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.ve_sysregs**

Type: `VE_SysRegs`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.vedcc**

Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.virtio_net**

VirtioNet device over MMIO transport.
Type: `VirtioNetMMIO`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.virtio_net_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.virtio_rng**

VirtioEntropy device over MMIO transport.
Type: `VirtioEntropyMMIO`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.virtioblockdevice**

VirtioBlock device over MMIO transport.
Type: `VirtioBlockMMIO`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.virtioblockdevice_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.virtiop9device**

virtio P9 server.
Type: `VirtioP9Device`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.virtiop9device_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.vis**

Display window for VE using Visualisation library.
Type: `VEVisualisation`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.bp.vram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.cci400**

Cache Coherent Interconnect for AXI4 ACE.
Type: `CCI400`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.clockdivider0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.cluster0**
ARM Cortex-A710 CT heterogeneous cluster model.
Type: `Cluster_ARM_CortexA710_Heterogeneous_Cluster`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.cluster0.DSU.shared_cache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.cluster0.subcluster0**
ARM Cortex-A510 CT Model in Heterogeneous SubCluster.
Type: `Subcluster_ARM_Cortex-A510`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.cluster0.subcluster0.cpu0**
ARM Cortex-A510 CT Model in Heterogeneous Cluster.
Type: `Cluster_ARM_Cortex-A510`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.cluster0.subcluster0.cpu0.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.cluster0.subcluster0.cpu0.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.cluster0.subcluster0.cpu0.l2cache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.cluster0.subcluster0.cpu1**
ARM Cortex-A510 CT Model in Heterogeneous Cluster.
Type: `Cluster_ARM_Cortex-A510`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.cluster0.subcluster0.cpu1.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.cluster0.subcluster0.cpu1.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.cluster0.subcluster0.cpu1.l2cache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.cluster0.subcluster0.cpu2**
ARM Cortex-A510 CT Model in Heterogeneous Cluster.
Type: `Cluster_ARM_Cortex-A510`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.cluster0.subcluster0.cpu2.l1dcache**
PV Cache.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `PVCache`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.cluster0.subcluster0.cpu2.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.cluster0.subcluster0.cpu2.l2cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.cluster0.subcluster0.cpu3**

ARM Cortex-A510 CT Model in Heterogeneous Cluster.
Type: `Cluster_ARM_Cortex-A510`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.cluster0.subcluster0.cpu3.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.cluster0.subcluster0.cpu3.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.cluster0.subcluster0.cpu3.l2cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.cluster0.subcluster1**

ARM Cortex-A710 Cluster CT model.
Type: `Subcluster_ARM_CortexA710`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.cluster0.subcluster1.cpu0**

ARM Cortex-A710 CT Model in Heterogeneous Cluster.
Type: `Cluster_ARM_CortexA710`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.cluster0.subcluster1.cpu0.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.cluster0.subcluster1.cpu0.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.cluster0.subcluster1.cpu0.l2cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.cluster0.subcluster1.cpu1**

ARM Cortex-A710 CT Model in Heterogeneous Cluster.
Type: `Cluster_ARM_CortexA710`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.cluster0.subcluster1.cpu1.l1dcache**

PV Cache.
Type: `PVCache`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A510x4_Cortex_A710x4.cluster0.subcluster1.cpu1.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.cluster0.subcluster1.cpu1.l2cache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.cluster0.subcluster1.cpu2**
ARM Cortex-A710 CT Model in Heterogeneous Cluster.
Type: `Cluster_ARM_CortexA710`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.cluster0.subcluster1.cpu2.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.cluster0.subcluster1.cpu2.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.cluster0.subcluster1.cpu2.l2cache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.cluster0.subcluster1.cpu3**
ARM Cortex-A710 CT Model in Heterogeneous Cluster.
Type: `Cluster_ARM_CortexA710`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.cluster0.subcluster1.cpu3.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.cluster0.subcluster1.cpu3.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.cluster0.subcluster1.cpu3.l2cache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.cluster0_labeller**
Type: `Labeller`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.dapmemlogger**
Bus Logger.
Type: `PVBusLogger`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.elfloader**
ELF loader component.
Type: `ElfLoader`.

**FVP_Base_Cortex_A510x4_Cortex_A710x4.gic_distributor**
GIC Interrupt Redistribution Infrastucture component.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `GIC_IRI`.

**`FVP_Base_Cortex_A510x4_Cortex_A710x4.pctl`**

Base Platforms Power Controller.
Type: `Base_PowerController`.

**`FVP_Base_Cortex_A510x4_Cortex_A710x4.pctl.timer_reset`**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**`FVP_Base_Cortex_A510x4_Cortex_A710x4.pctl.timer_reset.timer`**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**`FVP_Base_Cortex_A510x4_Cortex_A710x4.pctl.timer_reset.timer.thread`**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**`FVP_Base_Cortex_A510x4_Cortex_A710x4.pctl.timer_reset.timer.thread_event`**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

## 13.15 FVP_Base_Cortex-A520

List of instances in FVP_Base_Cortex-A520.

### FVP_Base_Cortex-A520 instances

**`FVP_Base_Cortex_A520`**

Base Platform Compute Subsystem for ARMCortexA520CT.
Type: `FVP_Base_Cortex_A520`.

**`FVP_Base_Cortex_A520.bp`**

Peripherals and address map for the Base Platform.
Type: `BasePlatformPeripherals`.

**`FVP_Base_Cortex_A520.bp.Timer_0_1`**

ARM Dual-Timer Module(SP804).

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `SP804_Timer`.

**FVP_Base_Cortex_A520.bp.Timer_0_1.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A520.bp.Timer_0_1.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A520.bp.Timer_0_1.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A520.bp.Timer_0_1.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A520.bp.Timer_2_3**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Cortex_A520.bp.Timer_2_3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A520.bp.Timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A520.bp.Timer_2_3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A520.bp.Timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A520.bp.ap_refclk**

ARM Generic Timer.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `MemoryMappedGenericTimer`.

**FVP_Base_Cortex_A520.bp.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**FVP_Base_Cortex_A520.bp.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A520.bp.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A520.bp.clock300MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A520.bp.clock32KHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A520.bp.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A520.bp.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A520.bp.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A520.bp.clockdivider**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A520.bp.dmc**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A520.bp.dmc_phy**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A520.bp.dummy_local_dap_rom**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A520.bp.dummy_ram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A520.bp.dummy_usb**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A520.bp.exclusive_monitor**
Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_Base_Cortex_A520.bp.flash0**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A520.bp.flash1**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A520.bp.flashloader0**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A520.bp.flashloader1**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A520.bp.generic_watchdog**
ARM Generic Watchdog.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `MemoryMappedGenericWatchdog`.

**FVP_Base_Cortex_A520.bp.hdlcd0**

ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).
Type: `PL370_HDLCD`.

**FVP_Base_Cortex_A520.bp.hdlcd0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A520.bp.hdlcd0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A520.bp.hdlcd0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A520.bp.hdlcd0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A520.bp.hdlcd0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A520.bp.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_Base_Cortex_A520.bp.mmc**

Generic Multimedia Card.
Type: `MMC`.

**FVP_Base_Cortex_A520.bp.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A520.bp.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A520.bp.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A520.bp.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A520.bp.mpe**

Encrypt memory transactions for each encryption context with an independent key to prevent mismatch access.
Type: `PVMemoryProtectionEngine`.

**FVP_Base_Cortex_A520.bp.nontrustedrom**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A520.bp.nontrustedromloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A520.bp.ns_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A520.bp.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A520.bp.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A520.bp.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A520.bp.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A520.bp.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A520.bp.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A520.bp.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A520.bp.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A520.bp.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A520.bp.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A520.bp.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A520.bp.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A520.bp.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A520.bp.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A520.bp.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A520.bp.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A520.bp.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A520.bp.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A520.bp.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A520.bp.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A520.bp.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A520.bp.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A520.bp.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A520.bp.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A520.bp.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

**FVP_Base_Cortex_A520.bp.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal()

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A520.bp.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A520.bp.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A520.bp.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A520.bp.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_Base_Cortex_A520.bp.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A520.bp.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A520.bp.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A520.bp.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A520.bp.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A520.bp.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A520.bp.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A520.bp.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A520.bp.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_Base_Cortex_A520.bp.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

**FVP_Base_Cortex_A520.bp.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A520.bp.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A520.bp.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A520.bp.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A520.bp.pl111_clcd_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A520.bp.pl180_mci**

ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

**FVP_Base_Cortex_A520.bp.ps2keyboard**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

**FVP_Base_Cortex_A520.bp.ps2keyboard.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A520.bp.ps2keyboard.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A520.bp.ps2keyboard.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A520.bp.ps2keyboard.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A520.bp.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

**FVP_Base_Cortex_A520.bp.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A520.bp.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A520.bp.ps2mouse.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A520.bp.ps2mouse.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A520.bp.psram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A520.bp.refcounter**

Memory Mapped Counter Module for Generic Timers.
Type: `MemoryMappedCounterModule`.

**FVP_Base_Cortex_A520.bp.reset_or**

Or Gate.
Type: `OrGate`.

**FVP_Base_Cortex_A520.bp.rl_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A520.bp.rt_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A520.bp.s_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A520.bp.secureDRAM**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A520.bp.secureSRAM**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A520.bp.secureflash**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A520.bp.secureflashloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A520.bp.smsc_91c111**

SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_Base_Cortex_A520.bp.sp805_wdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A520.bp.sp805_wdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A520.bp.sp805_wdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A520.bp.sp805_wdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A520.bp.sp805_wdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A520.bp.sp810_sysctrl**

Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_Base_Cortex_A520.bp.sp810_sysctrl.clkdiv_clk0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A520.bp.sp810_sysctrl.clkdiv_clk1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A520.bp.sp810_sysctrl.clkdiv_clk2**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A520.bp.sp810_sysctrl.clkdiv_clk3**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A520.bp.sram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A520.bp.terminal_0**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A520.bp.terminal_1**

Telnet terminal interface.
Type: `TelnetTerminal`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A520.bp.terminal_2**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A520.bp.terminal_3**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A520.bp.trusted_key_storage**

Trusted Root-Key Storage unit.
Type: `RootKeyStorage`.

**FVP_Base_Cortex_A520.bp.trusted_nv_counter**

Trusted Non-Volatile Counter unit.
Type: `NonVolatileCounter`.

**FVP_Base_Cortex_A520.bp.trusted_rng**

Random Number Generator unit.
Type: `RandomNumberGenerator`.

**FVP_Base_Cortex_A520.bp.trusted_watchdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A520.bp.trusted_watchdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A520.bp.trusted_watchdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A520.bp.trusted_watchdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A520.bp.trusted_watchdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A520.bp.tzc_400**

TrustZone Address Space Controller.
Type: `TZC_400`.

**FVP_Base_Cortex_A520.bp.ve_sysregs**

Type: `VE_SysRegs`.

**FVP_Base_Cortex_A520.bp.vedcc**

Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

**FVP_Base_Cortex_A520.bp.virtio_net**

VirtioNet device over MMIO transport.
Type: `VirtioNetMMIO`.

**FVP_Base_Cortex_A520.bp.virtio_net_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A520.bp.virtio_rng**

VirtioEntropy device over MMIO transport.
Type: `VirtioEntropyMMIO`.

**FVP_Base_Cortex_A520.bp.virtioblockdevice**

VirtioBlock device over MMIO transport.
Type: `VirtioBlockMMIO`.

**FVP_Base_Cortex_A520.bp.virtioblockdevice_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A520.bp.virtiop9device**

virtio P9 server.
Type: `VirtioP9Device`.

**FVP_Base_Cortex_A520.bp.virtiop9device_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A520.bp.vis**

Display window for VE using Visualisation library.
Type: `VEVisualisation`.

**FVP_Base_Cortex_A520.bp.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_Base_Cortex_A520.bp.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A520.bp.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A520.bp.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A520.bp.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A520.bp.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A520.bp.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A520.bp.vram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A520.cci400**

Cache Coherent Interconnect for AXI4 ACE.
Type: `CCI400`.

**FVP_Base_Cortex_A520.clockdivider0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A520.cluster0**

ARM Cortex-A520 Cluster CT model.
Type: `Cluster_ARM_Cortex-A520`.

**FVP_Base_Cortex_A520.cluster0.DSU.shared_cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A520.cluster0.cpu0**

ARM Cortex-A520 CT model.
Type: `ARM_Cortex-A520`.

**FVP_Base_Cortex_A520.cluster0.cpu0.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A520.cluster0.cpu0.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A520.cluster0.cpu0.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A520.cluster0.cpu0.l2cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A520.cluster0.cpu1**

ARM Cortex-A520 CT model.
Type: `ARM_Cortex-A520`.

**FVP_Base_Cortex_A520.cluster0.cpu1.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A520.cluster0.cpu1.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A520.cluster0.cpu1.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A520.cluster0.cpu1.l2cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A520.cluster0.cpu10**

ARM Cortex-A520 CT model.
Type: `ARM_Cortex-A520`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A520.cluster0.cpu10.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A520.cluster0.cpu10.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A520.cluster0.cpu10.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A520.cluster0.cpu10.l2cache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A520.cluster0.cpu11**
ARM Cortex-A520 CT model.
Type: `ARM_Cortex-A520`.

**FVP_Base_Cortex_A520.cluster0.cpu11.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A520.cluster0.cpu11.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A520.cluster0.cpu11.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A520.cluster0.cpu11.l2cache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A520.cluster0.cpu12**
ARM Cortex-A520 CT model.
Type: `ARM_Cortex-A520`.

**FVP_Base_Cortex_A520.cluster0.cpu12.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A520.cluster0.cpu12.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A520.cluster0.cpu12.l1icache**
PV Cache.
Type: `PVCache`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A520.cluster0.cpu12.l2cache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A520.cluster0.cpu13**
ARM Cortex-A520 CT model.
Type: `ARM_Cortex-A520`.

**FVP_Base_Cortex_A520.cluster0.cpu13.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A520.cluster0.cpu13.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A520.cluster0.cpu13.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A520.cluster0.cpu13.l2cache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A520.cluster0.cpu2**
ARM Cortex-A520 CT model.
Type: `ARM_Cortex-A520`.

**FVP_Base_Cortex_A520.cluster0.cpu2.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A520.cluster0.cpu2.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A520.cluster0.cpu2.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A520.cluster0.cpu2.l2cache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A520.cluster0.cpu3**
ARM Cortex-A520 CT model.
Type: `ARM_Cortex-A520`.

**FVP_Base_Cortex_A520.cluster0.cpu3.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A520.cluster0.cpu3.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A520.cluster0.cpu3.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A520.cluster0.cpu3.l2cache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A520.cluster0.cpu4**
ARM Cortex-A520 CT model.
Type: `ARM_Cortex-A520`.

**FVP_Base_Cortex_A520.cluster0.cpu4.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A520.cluster0.cpu4.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A520.cluster0.cpu4.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A520.cluster0.cpu4.l2cache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A520.cluster0.cpu5**
ARM Cortex-A520 CT model.
Type: `ARM_Cortex-A520`.

**FVP_Base_Cortex_A520.cluster0.cpu5.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A520.cluster0.cpu5.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A520.cluster0.cpu5.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A520.cluster0.cpu5.l2cache**
PV Cache.
Type: `PVCache`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A520.cluster0.cpu6**
ARM Cortex-A520 CT model.
Type: `ARM_Cortex-A520`.

**FVP_Base_Cortex_A520.cluster0.cpu6.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A520.cluster0.cpu6.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A520.cluster0.cpu6.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A520.cluster0.cpu6.l2cache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A520.cluster0.cpu7**
ARM Cortex-A520 CT model.
Type: `ARM_Cortex-A520`.

**FVP_Base_Cortex_A520.cluster0.cpu7.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A520.cluster0.cpu7.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A520.cluster0.cpu7.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A520.cluster0.cpu7.l2cache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A520.cluster0.cpu8**
ARM Cortex-A520 CT model.
Type: `ARM_Cortex-A520`.

**FVP_Base_Cortex_A520.cluster0.cpu8.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A520.cluster0.cpu8.l1dcache**
PV Cache.
Type: `PVCache`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A520.cluster0.cpu8.l1icache**
PV Cache.
Type: PVCache.

**FVP_Base_Cortex_A520.cluster0.cpu8.l2cache**
PV Cache.
Type: PVCache.

**FVP_Base_Cortex_A520.cluster0.cpu9**
ARM Cortex-A520 CT model.
Type: ARM_Cortex-A520.

**FVP_Base_Cortex_A520.cluster0.cpu9.dtlb**
TLB - instruction, data or unified.
Type: TlbCadi.

**FVP_Base_Cortex_A520.cluster0.cpu9.l1dcache**
PV Cache.
Type: PVCache.

**FVP_Base_Cortex_A520.cluster0.cpu9.l1icache**
PV Cache.
Type: PVCache.

**FVP_Base_Cortex_A520.cluster0.cpu9.l2cache**
PV Cache.
Type: PVCache.

**FVP_Base_Cortex_A520.cluster0_labeller**
Type: Labeller.

**FVP_Base_Cortex_A520.dapmemlogger**
Bus Logger.
Type: PVBusLogger.

**FVP_Base_Cortex_A520.elfloader**
ELF loader component.
Type: ElfLoader.

**FVP_Base_Cortex_A520.gic_distributor**
GIC Interrupt Redistribution Infrastucture component.
Type: GIC_IRI.

**FVP_Base_Cortex_A520.pctl**
Base Platforms Power Controller.
Type: Base_PowerController.

**FVP_Base_Cortex_A520.pctl.timer_reset**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**`FVP_Base_Cortex_A520.pctl.timer_reset.timer`**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**`FVP_Base_Cortex_A520.pctl.timer_reset.timer.thread`**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**`FVP_Base_Cortex_A520.pctl.timer_reset.timer.thread_event`**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

## 13.16 FVP_Base_Cortex-A53x1

List of instances in FVP_Base_Cortex-A53x1.

### FVP_Base_Cortex-A53x1 instances

**`FVP_Base_Cortex_A53x1`**

Base Platform Compute Subsystem for ARMCortexA53x1CT.
Type: `FVP_Base_Cortex_A53x1`.

**`FVP_Base_Cortex_A53x1.bp`**

Peripherals and address map for the Base Platform.
Type: `BasePlatformPeripherals`.

**`FVP_Base_Cortex_A53x1.bp.Timer_0_1`**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**`FVP_Base_Cortex_A53x1.bp.Timer_0_1.clk_div0`**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**`FVP_Base_Cortex_A53x1.bp.Timer_0_1.clk_div1`**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A53x1.bp.Timer_0_1.counter0**
Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A53x1.bp.Timer_0_1.counter1**
Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A53x1.bp.Timer_2_3**
ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Cortex_A53x1.bp.Timer_2_3.clk_div0**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A53x1.bp.Timer_2_3.clk_div1**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A53x1.bp.Timer_2_3.counter0**
Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A53x1.bp.Timer_2_3.counter1**
Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A53x1.bp.ap_refclk**
ARM Generic Timer.
Type: `MemoryMappedGenericTimer`.

**FVP_Base_Cortex_A53x1.bp.audioout**
SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**FVP_Base_Cortex_A53x1.bp.clock100Hz**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A53x1.bp.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A53x1.bp.clock300MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A53x1.bp.clock32KHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A53x1.bp.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A53x1.bp.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A53x1.bp.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A53x1.bp.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A53x1.bp.dmc**

RAM device, can be dynamic or static ram.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `RAMDevice`.

**FVP_Base_Cortex_A53x1.bp.dmc_phy**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A53x1.bp.dummy_local_dap_rom**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A53x1.bp.dummy_ram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A53x1.bp.dummy_usb**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A53x1.bp.exclusive_monitor**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_Base_Cortex_A53x1.bp.flash0**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A53x1.bp.flash1**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A53x1.bp.flashloader0**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A53x1.bp.flashloader1**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A53x1.bp.generic_watchdog**

ARM Generic Watchdog.
Type: `MemoryMappedGenericWatchdog`.

**FVP_Base_Cortex_A53x1.bp.hdlcd0**

ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).
Type: `PL370_HDLCD`.

**FVP_Base_Cortex_A53x1.bp.hdlcd0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A53x1.bp.hdlcd0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A53x1.bp.hdlcd0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A53x1.bp.hdlcd0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A53x1.bp.hdlcd0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A53x1.bp.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_Base_Cortex_A53x1.bp.mmc**

Generic Multimedia Card.
Type: `MMC`.

**FVP_Base_Cortex_A53x1.bp.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A53x1.bp.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A53x1.bp.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A53x1.bp.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A53x1.bp.mpe**

Encrypt memory transactions for each encryption context with an independent key to prevent mismatch access.
Type: `PVMemoryProtectionEngine`.

**FVP_Base_Cortex_A53x1.bp.nontrustedrom**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A53x1.bp.nontrustedromloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A53x1.bp.ns_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A53x1.bp.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A53x1.bp.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A53x1.bp.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A53x1.bp.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A53x1.bp.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A53x1.bp.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A53x1.bp.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A53x1.bp.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A53x1.bp.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A53x1.bp.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A53x1.bp.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A53x1.bp.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A53x1.bp.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A53x1.bp.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A53x1.bp.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A53x1.bp.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A53x1.bp.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A53x1.bp.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A53x1.bp.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A53x1.bp.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A53x1.bp.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A53x1.bp.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A53x1.bp.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A53x1.bp.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A53x1.bp.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

**FVP_Base_Cortex_A53x1.bp.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A53x1.bp.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A53x1.bp.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A53x1.bp.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A53x1.bp.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_Base_Cortex_A53x1.bp.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A53x1.bp.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A53x1.bp.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A53x1.bp.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A53x1.bp.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A53x1.bp.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A53x1.bp.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A53x1.bp.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A53x1.bp.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_Base_Cortex_A53x1.bp.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

**FVP_Base_Cortex_A53x1.bp.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A53x1.bp.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A53x1.bp.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A53x1.bp.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A53x1.bp.pl111_clcd_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A53x1.bp.pl180_mci**

ARM PrimeCell Multimedia Card Interface (PL180).

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `PL180_MCI`.

**FVP_Base_Cortex_A53x1.bp.ps2keyboard**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

**FVP_Base_Cortex_A53x1.bp.ps2keyboard.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A53x1.bp.ps2keyboard.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A53x1.bp.ps2keyboard.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A53x1.bp.ps2keyboard.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A53x1.bp.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

**FVP_Base_Cortex_A53x1.bp.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A53x1.bp.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A53x1.bp.ps2mouse.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A53x1.bp.ps2mouse.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A53x1.bp.psram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A53x1.bp.refcounter**

Memory Mapped Counter Module for Generic Timers.
Type: `MemoryMappedCounterModule`.

**FVP_Base_Cortex_A53x1.bp.reset_or**

Or Gate.
Type: `OrGate`.

**FVP_Base_Cortex_A53x1.bp.rl_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A53x1.bp.rt_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A53x1.bp.s_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A53x1.bp.secureDRAM**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A53x1.bp.secureSRAM**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A53x1.bp.secureflash**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A53x1.bp.secureflashloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A53x1.bp.smsc_91c111**

SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_Base_Cortex_A53x1.bp.sp805_wdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A53x1.bp.sp805_wdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A53x1.bp.sp805_wdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A53x1.bp.sp805_wdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A53x1.bp.sp805_wdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A53x1.bp.sp810_sysctrl**

Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_Base_Cortex_A53x1.bp.sp810_sysctrl.clkdiv_clk0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A53x1.bp.sp810_sysctrl.clkdiv_clk1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A53x1.bp.sp810_sysctrl.clkdiv_clk2**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A53x1.bp.sp810_sysctrl.clkdiv_clk3**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A53x1.bp.sram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A53x1.bp.terminal_0**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A53x1.bp.terminal_1**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A53x1.bp.terminal_2**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A53x1.bp.terminal_3**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A53x1.bp.trusted_key_storage**

Trusted Root-Key Storage unit.
Type: `RootKeyStorage`.

**FVP_Base_Cortex_A53x1.bp.trusted_nv_counter**

Trusted Non-Volatile Counter unit.
Type: `NonVolatileCounter`.

**FVP_Base_Cortex_A53x1.bp.trusted_rng**

Random Number Generator unit.
Type: `RandomNumberGenerator`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A53x1.bp.trusted_watchdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A53x1.bp.trusted_watchdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A53x1.bp.trusted_watchdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A53x1.bp.trusted_watchdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A53x1.bp.trusted_watchdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A53x1.bp.tzc_400**

TrustZone Address Space Controller.
Type: `TZC_400`.

**FVP_Base_Cortex_A53x1.bp.ve_sysregs**

Type: `VE_SysRegs`.

**FVP_Base_Cortex_A53x1.bp.vedcc**

Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

**FVP_Base_Cortex_A53x1.bp.virtio_net**

VirtioNet device over MMIO transport.
Type: `VirtioNetMMIO`.

**FVP_Base_Cortex_A53x1.bp.virtio_net_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A53x1.bp.virtio_rng**

VirtioEntropy device over MMIO transport.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `VirtioEntropyMMIO`.

**FVP_Base_Cortex_A53x1.bp.virtioblockdevice**

VirtioBlock device over MMIO transport.
Type: `VirtioBlockMMIO`.

**FVP_Base_Cortex_A53x1.bp.virtioblockdevice_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A53x1.bp.virtiop9device**

virtio P9 server.
Type: `VirtioP9Device`.

**FVP_Base_Cortex_A53x1.bp.virtiop9device_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A53x1.bp.vis**

Display window for VE using Visualisation library.
Type: `VEVisualisation`.

**FVP_Base_Cortex_A53x1.bp.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_Base_Cortex_A53x1.bp.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A53x1.bp.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A53x1.bp.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A53x1.bp.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A53x1.bp.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A53x1.bp.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A53x1.bp.vram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A53x1.cci400**

Cache Coherent Interconnect for AXI4 ACE.
Type: `CCI400`.

**FVP_Base_Cortex_A53x1.clockdivider0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A53x1.cluster0**

ARM Cortex-A53 Cluster CT model.
Type: `Cluster_ARM_Cortex-A53`.

**FVP_Base_Cortex_A53x1.cluster0.cpu0**

ARM Cortex-A53 CT model.
Type: `ARM_Cortex-A53`.

**FVP_Base_Cortex_A53x1.cluster0.cpu0.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A53x1.cluster0.cpu0.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A53x1.cluster0.cpu0.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A53x1.cluster0.l2_cache**

PV Cache.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `PVCache`.

**FVP_Base_Cortex_A53x1.cluster0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A53x1.dapmemlogger**

Bus Logger.
Type: `PVBusLogger`.

**FVP_Base_Cortex_A53x1.elfloader**

ELF loader component.
Type: `ElfLoader`.

**FVP_Base_Cortex_A53x1.gic_distributor**

GIC Interrupt Redistribution Infrastucture component.
Type: `GIC_IRI`.

**FVP_Base_Cortex_A53x1.pctl**

Base Platforms Power Controller.
Type: `Base_PowerController`.

**FVP_Base_Cortex_A53x1.pctl.timer_reset**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A53x1.pctl.timer_reset.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A53x1.pctl.timer_reset.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A53x1.pctl.timer_reset.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

# 13.17 FVP_Base_Cortex-A53x2

List of instances in FVP_Base_Cortex-A53x2.

## FVP_Base_Cortex-A53x2 instances

**FVP_Base_Cortex_A53x2**

Base Platform Compute Subsystem for ARMCortexA53x2CT.
Type: `FVP_Base_Cortex_A53x2`.

**FVP_Base_Cortex_A53x2.bp**

Peripherals and address map for the Base Platform.
Type: `BasePlatformPeripherals`.

**FVP_Base_Cortex_A53x2.bp.Timer_0_1**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Cortex_A53x2.bp.Timer_0_1.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A53x2.bp.Timer_0_1.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A53x2.bp.Timer_0_1.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A53x2.bp.Timer_0_1.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A53x2.bp.Timer_2_3**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Cortex_A53x2.bp.Timer_2_3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A53x2.bp.Timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A53x2.bp.Timer_2_3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A53x2.bp.Timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A53x2.bp.ap_refclk**

ARM Generic Timer.
Type: `MemoryMappedGenericTimer`.

**FVP_Base_Cortex_A53x2.bp.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**FVP_Base_Cortex_A53x2.bp.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A53x2.bp.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A53x2.bp.clock300MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A53x2.bp.clock32KHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A53x2.bp.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A53x2.bp.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A53x2.bp.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A53x2.bp.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A53x2.bp.dmc**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A53x2.bp.dmc_phy**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A53x2.bp.dummy_local_dap_rom**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A53x2.bp.dummy_ram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A53x2.bp.dummy_usb**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A53x2.bp.exclusive_monitor**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A53x2.bp.flash0**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A53x2.bp.flash1**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A53x2.bp.flashloader0**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A53x2.bp.flashloader1**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A53x2.bp.generic_watchdog**

ARM Generic Watchdog.
Type: `MemoryMappedGenericWatchdog`.

**FVP_Base_Cortex_A53x2.bp.hdlcd0**

ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).
Type: `PL370_HDLCD`.

**FVP_Base_Cortex_A53x2.bp.hdlcd0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A53x2.bp.hdlcd0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A53x2.bp.hdlcd0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A53x2.bp.hdlcd0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A53x2.bp.hdlcd0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A53x2.bp.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_Base_Cortex_A53x2.bp.mmc**

Generic Multimedia Card.
Type: `MMC`.

**FVP_Base_Cortex_A53x2.bp.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A53x2.bp.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A53x2.bp.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A53x2.bp.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A53x2.bp.mpe**

Encrypt memory transactions for each encryption context with an independent key to prevent mismatch access.
Type: `PVMemoryProtectionEngine`.

**FVP_Base_Cortex_A53x2.bp.nontrustedrom**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A53x2.bp.nontrustedromloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A53x2.bp.ns_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A53x2.bp.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A53x2.bp.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A53x2.bp.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A53x2.bp.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A53x2.bp.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A53x2.bp.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A53x2.bp.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A53x2.bp.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A53x2.bp.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A53x2.bp.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A53x2.bp.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A53x2.bp.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A53x2.bp.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A53x2.bp.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A53x2.bp.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ClockTimerThread`.

**FVP_Base_Cortex_A53x2.bp.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A53x2.bp.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A53x2.bp.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A53x2.bp.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A53x2.bp.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A53x2.bp.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A53x2.bp.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A53x2.bp.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A53x2.bp.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A53x2.bp.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

**FVP_Base_Cortex_A53x2.bp.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A53x2.bp.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A53x2.bp.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A53x2.bp.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A53x2.bp.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_Base_Cortex_A53x2.bp.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A53x2.bp.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A53x2.bp.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A53x2.bp.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A53x2.bp.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A53x2.bp.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A53x2.bp.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A53x2.bp.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A53x2.bp.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_Base_Cortex_A53x2.bp.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A53x2.bp.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A53x2.bp.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A53x2.bp.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A53x2.bp.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A53x2.bp.pl111_clcd_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A53x2.bp.pl180_mci**

ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

**FVP_Base_Cortex_A53x2.bp.ps2keyboard**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

**FVP_Base_Cortex_A53x2.bp.ps2keyboard.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A53x2.bp.ps2keyboard.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A53x2.bp.ps2keyboard.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A53x2.bp.ps2keyboard.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A53x2.bp.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

**FVP_Base_Cortex_A53x2.bp.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A53x2.bp.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A53x2.bp.ps2mouse.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A53x2.bp.ps2mouse.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A53x2.bp.psram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A53x2.bp.refcounter**
Memory Mapped Counter Module for Generic Timers.
Type: `MemoryMappedCounterModule`.

**FVP_Base_Cortex_A53x2.bp.reset_or**
Or Gate.
Type: `OrGate`.

**FVP_Base_Cortex_A53x2.bp.rl_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A53x2.bp.rt_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A53x2.bp.s_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A53x2.bp.secureDRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A53x2.bp.secureSRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A53x2.bp.secureflash**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A53x2.bp.secureflashloader**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A53x2.bp.smsc_91c111**
SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_Base_Cortex_A53x2.bp.sp805_wdog**
ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A53x2.bp.sp805_wdog.clocktimer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A53x2.bp.sp805_wdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A53x2.bp.sp805_wdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A53x2.bp.sp805_wdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A53x2.bp.sp810_sysctrl**

Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_Base_Cortex_A53x2.bp.sp810_sysctrl.clkdiv_clk0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A53x2.bp.sp810_sysctrl.clkdiv_clk1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A53x2.bp.sp810_sysctrl.clkdiv_clk2**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A53x2.bp.sp810_sysctrl.clkdiv_clk3**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A53x2.bp.sram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A53x2.bp.terminal_0**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A53x2.bp.terminal_1**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A53x2.bp.terminal_2**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A53x2.bp.terminal_3**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A53x2.bp.trusted_key_storage**
Trusted Root-Key Storage unit.
Type: `RootKeyStorage`.

**FVP_Base_Cortex_A53x2.bp.trusted_nv_counter**
Trusted Non-Volatile Counter unit.
Type: `NonVolatileCounter`.

**FVP_Base_Cortex_A53x2.bp.trusted_rng**
Random Number Generator unit.
Type: `RandomNumberGenerator`.

**FVP_Base_Cortex_A53x2.bp.trusted_watchdog**
ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A53x2.bp.trusted_watchdog.clocktimer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A53x2.bp.trusted_watchdog.clocktimer.timer**
A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A53x2.bp.trusted_watchdog.clocktimer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A53x2.bp.trusted_watchdog.clocktimer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A53x2.bp.tzc_400**
TrustZone Address Space Controller.
Type: `TZC_400`.

**FVP_Base_Cortex_A53x2.bp.ve_sysregs**
Type: `VE_SysRegs`.

**FVP_Base_Cortex_A53x2.bp.vedcc**
Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

**FVP_Base_Cortex_A53x2.bp.virtio_net**
VirtioNet device over MMIO transport.
Type: `VirtioNetMMIO`.

**FVP_Base_Cortex_A53x2.bp.virtio_net_labeller**
Type: `Labeller`.

**FVP_Base_Cortex_A53x2.bp.virtio_rng**
VirtioEntropy device over MMIO transport.
Type: `VirtioEntropyMMIO`.

**FVP_Base_Cortex_A53x2.bp.virtioblockdevice**
VirtioBlock device over MMIO transport.
Type: `VirtioBlockMMIO`.

**FVP_Base_Cortex_A53x2.bp.virtioblockdevice_labeller**
Type: `Labeller`.

**FVP_Base_Cortex_A53x2.bp.virtiop9device**
virtio P9 server.
Type: `VirtioP9Device`.

**FVP_Base_Cortex_A53x2.bp.virtiop9device_labeller**
Type: `Labeller`.

**FVP_Base_Cortex_A53x2.bp.vis**
Display window for VE using Visualisation library.
Type: `VEVisualisation`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A53x2.bp.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_Base_Cortex_A53x2.bp.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A53x2.bp.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A53x2.bp.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A53x2.bp.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A53x2.bp.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A53x2.bp.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A53x2.bp.vram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A53x2.cci400**

Cache Coherent Interconnect for AXI4 ACE.
Type: `CCI400`.

**FVP_Base_Cortex_A53x2.clockdivider0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A53x2.cluster0**

ARM Cortex-A53 Cluster CT model.
Type: `Cluster_ARM_Cortex-A53`.

**FVP_Base_Cortex_A53x2.cluster0.cpu0**

ARM Cortex-A53 CT model.
Type: `ARM_Cortex-A53`.

**FVP_Base_Cortex_A53x2.cluster0.cpu0.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A53x2.cluster0.cpu0.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A53x2.cluster0.cpu0.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A53x2.cluster0.cpu1**

ARM Cortex-A53 CT model.
Type: `ARM_Cortex-A53`.

**FVP_Base_Cortex_A53x2.cluster0.cpu1.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A53x2.cluster0.cpu1.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A53x2.cluster0.cpu1.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A53x2.cluster0.l2_cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A53x2.cluster0_labeller**

Type: `Labeller`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A53x2.dapmemlogger**

Bus Logger.
Type: `PVBusLogger`.

**FVP_Base_Cortex_A53x2.elfloader**

ELF loader component.
Type: `ElfLoader`.

**FVP_Base_Cortex_A53x2.gic_distributor**

GIC Interrupt Redistribution Infrastucture component.
Type: `GIC_IRI`.

**FVP_Base_Cortex_A53x2.pctl**

Base Platforms Power Controller.
Type: `Base_PowerController`.

**FVP_Base_Cortex_A53x2.pctl.timer_reset**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A53x2.pctl.timer_reset.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A53x2.pctl.timer_reset.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A53x2.pctl.timer_reset.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

# 13.18 FVP_Base_Cortex-A53x4

List of instances in FVP_Base_Cortex-A53x4.

## FVP_Base_Cortex-A53x4 instances

**FVP_Base_Cortex_A53x4**

Base Platform Compute Subsystem for ARMCortexA53x4CT.
Type: `FVP_Base_Cortex_A53x4`.

**FVP_Base_Cortex_A53x4.bp**

Peripherals and address map for the Base Platform.
Type: `BasePlatformPeripherals`.

**FVP_Base_Cortex_A53x4.bp.Timer_0_1**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Cortex_A53x4.bp.Timer_0_1.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A53x4.bp.Timer_0_1.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A53x4.bp.Timer_0_1.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A53x4.bp.Timer_0_1.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A53x4.bp.Timer_2_3**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Cortex_A53x4.bp.Timer_2_3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A53x4.bp.Timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A53x4.bp.Timer_2_3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A53x4.bp.Timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A53x4.bp.ap_refclk**

ARM Generic Timer.
Type: `MemoryMappedGenericTimer`.

**FVP_Base_Cortex_A53x4.bp.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**FVP_Base_Cortex_A53x4.bp.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A53x4.bp.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A53x4.bp.clock300MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A53x4.bp.clock32KHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A53x4.bp.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A53x4.bp.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A53x4.bp.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A53x4.bp.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A53x4.bp.dmc**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A53x4.bp.dmc_phy**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A53x4.bp.dummy_local_dap_rom**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A53x4.bp.dummy_ram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A53x4.bp.dummy_usb**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A53x4.bp.exclusive_monitor**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A53x4.bp.flash0**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A53x4.bp.flash1**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A53x4.bp.flashloader0**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A53x4.bp.flashloader1**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A53x4.bp.generic_watchdog**

ARM Generic Watchdog.
Type: `MemoryMappedGenericWatchdog`.

**FVP_Base_Cortex_A53x4.bp.hdlcd0**

ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).
Type: `PL370_HDLCD`.

**FVP_Base_Cortex_A53x4.bp.hdlcd0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A53x4.bp.hdlcd0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A53x4.bp.hdlcd0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A53x4.bp.hdlcd0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A53x4.bp.hdlcd0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A53x4.bp.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_Base_Cortex_A53x4.bp.mmc**

Generic Multimedia Card.
Type: `MMC`.

**FVP_Base_Cortex_A53x4.bp.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A53x4.bp.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A53x4.bp.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A53x4.bp.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A53x4.bp.mpe**

Encrypt memory transactions for each encryption context with an independent key to prevent mismatch access.
Type: `PVMemoryProtectionEngine`.

**FVP_Base_Cortex_A53x4.bp.nontrustedrom**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A53x4.bp.nontrustedromloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A53x4.bp.ns_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A53x4.bp.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A53x4.bp.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A53x4.bp.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A53x4.bp.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A53x4.bp.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A53x4.bp.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A53x4.bp.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A53x4.bp.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A53x4.bp.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A53x4.bp.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A53x4.bp.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A53x4.bp.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A53x4.bp.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A53x4.bp.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A53x4.bp.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ClockTimerThread`.

**FVP_Base_Cortex_A53x4.bp.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A53x4.bp.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A53x4.bp.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A53x4.bp.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A53x4.bp.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A53x4.bp.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A53x4.bp.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A53x4.bp.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A53x4.bp.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A53x4.bp.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

**FVP_Base_Cortex_A53x4.bp.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A53x4.bp.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A53x4.bp.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A53x4.bp.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A53x4.bp.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_Base_Cortex_A53x4.bp.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A53x4.bp.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A53x4.bp.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A53x4.bp.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A53x4.bp.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A53x4.bp.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A53x4.bp.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A53x4.bp.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A53x4.bp.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_Base_Cortex_A53x4.bp.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A53x4.bp.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A53x4.bp.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A53x4.bp.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A53x4.bp.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A53x4.bp.pl111_clcd_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A53x4.bp.pl180_mci**

ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

**FVP_Base_Cortex_A53x4.bp.ps2keyboard**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

**FVP_Base_Cortex_A53x4.bp.ps2keyboard.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A53x4.bp.ps2keyboard.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A53x4.bp.ps2keyboard.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A53x4.bp.ps2keyboard.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A53x4.bp.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

**FVP_Base_Cortex_A53x4.bp.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A53x4.bp.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A53x4.bp.ps2mouse.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A53x4.bp.ps2mouse.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A53x4.bp.psram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A53x4.bp.refcounter**
Memory Mapped Counter Module for Generic Timers.
Type: `MemoryMappedCounterModule`.

**FVP_Base_Cortex_A53x4.bp.reset_or**
Or Gate.
Type: `OrGate`.

**FVP_Base_Cortex_A53x4.bp.rl_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A53x4.bp.rt_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A53x4.bp.s_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A53x4.bp.secureDRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A53x4.bp.secureSRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A53x4.bp.secureflash**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A53x4.bp.secureflashloader**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A53x4.bp.smsc_91c111**
SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_Base_Cortex_A53x4.bp.sp805_wdog**
ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A53x4.bp.sp805_wdog.clocktimer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A53x4.bp.sp805_wdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A53x4.bp.sp805_wdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A53x4.bp.sp805_wdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A53x4.bp.sp810_sysctrl**

Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_Base_Cortex_A53x4.bp.sp810_sysctrl.clkdiv_clk0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A53x4.bp.sp810_sysctrl.clkdiv_clk1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A53x4.bp.sp810_sysctrl.clkdiv_clk2**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A53x4.bp.sp810_sysctrl.clkdiv_clk3**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A53x4.bp.sram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A53x4.bp.terminal_0**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A53x4.bp.terminal_1**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A53x4.bp.terminal_2**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A53x4.bp.terminal_3**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A53x4.bp.trusted_key_storage**
Trusted Root-Key Storage unit.
Type: `RootKeyStorage`.

**FVP_Base_Cortex_A53x4.bp.trusted_nv_counter**
Trusted Non-Volatile Counter unit.
Type: `NonVolatileCounter`.

**FVP_Base_Cortex_A53x4.bp.trusted_rng**
Random Number Generator unit.
Type: `RandomNumberGenerator`.

**FVP_Base_Cortex_A53x4.bp.trusted_watchdog**
ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A53x4.bp.trusted_watchdog.clocktimer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A53x4.bp.trusted_watchdog.clocktimer.timer**
A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A53x4.bp.trusted_watchdog.clocktimer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A53x4.bp.trusted_watchdog.clocktimer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A53x4.bp.tzc_400**
TrustZone Address Space Controller.
Type: `TZC_400`.

**FVP_Base_Cortex_A53x4.bp.ve_sysregs**
Type: `VE_SysRegs`.

**FVP_Base_Cortex_A53x4.bp.vedcc**
Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

**FVP_Base_Cortex_A53x4.bp.virtio_net**
VirtioNet device over MMIO transport.
Type: `VirtioNetMMIO`.

**FVP_Base_Cortex_A53x4.bp.virtio_net_labeller**
Type: `Labeller`.

**FVP_Base_Cortex_A53x4.bp.virtio_rng**
VirtioEntropy device over MMIO transport.
Type: `VirtioEntropyMMIO`.

**FVP_Base_Cortex_A53x4.bp.virtioblockdevice**
VirtioBlock device over MMIO transport.
Type: `VirtioBlockMMIO`.

**FVP_Base_Cortex_A53x4.bp.virtioblockdevice_labeller**
Type: `Labeller`.

**FVP_Base_Cortex_A53x4.bp.virtiop9device**
virtio P9 server.
Type: `VirtioP9Device`.

**FVP_Base_Cortex_A53x4.bp.virtiop9device_labeller**
Type: `Labeller`.

**FVP_Base_Cortex_A53x4.bp.vis**
Display window for VE using Visualisation library.
Type: `VEVisualisation`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A53x4.bp.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_Base_Cortex_A53x4.bp.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A53x4.bp.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A53x4.bp.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A53x4.bp.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A53x4.bp.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A53x4.bp.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A53x4.bp.vram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A53x4.cci400**

Cache Coherent Interconnect for AXI4 ACE.
Type: `CCI400`.

**FVP_Base_Cortex_A53x4.clockdivider0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A53x4.cluster0**

ARM Cortex-A53 Cluster CT model.
Type: `Cluster_ARM_Cortex-A53`.

**FVP_Base_Cortex_A53x4.cluster0.cpu0**

ARM Cortex-A53 CT model.
Type: `ARM_Cortex-A53`.

**FVP_Base_Cortex_A53x4.cluster0.cpu0.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A53x4.cluster0.cpu0.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A53x4.cluster0.cpu0.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A53x4.cluster0.cpu1**

ARM Cortex-A53 CT model.
Type: `ARM_Cortex-A53`.

**FVP_Base_Cortex_A53x4.cluster0.cpu1.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A53x4.cluster0.cpu1.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A53x4.cluster0.cpu1.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A53x4.cluster0.cpu2**

ARM Cortex-A53 CT model.
Type: `ARM_Cortex-A53`.

**FVP_Base_Cortex_A53x4.cluster0.cpu2.dtlb**

TLB - instruction, data or unified.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `TlbCadi`.

**FVP_Base_Cortex_A53x4.cluster0.cpu2.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A53x4.cluster0.cpu2.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A53x4.cluster0.cpu3**

ARM Cortex-A53 CT model.
Type: `ARM_Cortex-A53`.

**FVP_Base_Cortex_A53x4.cluster0.cpu3.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A53x4.cluster0.cpu3.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A53x4.cluster0.cpu3.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A53x4.cluster0.l2_cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A53x4.cluster0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A53x4.dapmemlogger**

Bus Logger.
Type: `PVBusLogger`.

**FVP_Base_Cortex_A53x4.elfloader**

ELF loader component.
Type: `ElfLoader`.

**FVP_Base_Cortex_A53x4.gic_distributor**

GIC Interrupt Redistribution Infrastucture component.
Type: `GIC_IRI`.

**FVP_Base_Cortex_A53x4.pctl**

Base Platforms Power Controller.
Type: `Base_PowerController`.

**FVP_Base_Cortex_A53x4.pctl.timer_reset**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**`FVP_Base_Cortex_A53x4.pctl.timer_reset.timer`**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**`FVP_Base_Cortex_A53x4.pctl.timer_reset.timer.thread`**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**`FVP_Base_Cortex_A53x4.pctl.timer_reset.timer.thread_event`**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

## 13.19 FVP_Base_Cortex-A55

List of instances in FVP_Base_Cortex-A55.

### FVP_Base_Cortex-A55 instances

**`FVP_Base_Cortex_A55`**

Base Platform Compute Subsystem for ARMCortexA55CT.
Type: `FVP_Base_Cortex_A55`.

**`FVP_Base_Cortex_A55.bp`**

Peripherals and address map for the Base Platform.
Type: `BasePlatformPeripherals`.

**`FVP_Base_Cortex_A55.bp.Timer_0_1`**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**`FVP_Base_Cortex_A55.bp.Timer_0_1.clk_div0`**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A55.bp.Timer_0_1.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55.bp.Timer_0_1.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A55.bp.Timer_0_1.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A55.bp.Timer_2_3**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Cortex_A55.bp.Timer_2_3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55.bp.Timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55.bp.Timer_2_3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A55.bp.Timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A55.bp.ap_refclk**

ARM Generic Timer.
Type: `MemoryMappedGenericTimer`.

**FVP_Base_Cortex_A55.bp.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**FVP_Base_Cortex_A55.bp.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55.bp.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55.bp.clock300MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55.bp.clock32KHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55.bp.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55.bp.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55.bp.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55.bp.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ClockDivider`.

**FVP_Base_Cortex_A55.bp.dmc**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55.bp.dmc_phy**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55.bp.dummy_local_dap_rom**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55.bp.dummy_ram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55.bp.dummy_usb**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55.bp.exclusive_monitor**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_Base_Cortex_A55.bp.flash0**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A55.bp.flash1**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A55.bp.flashloader0**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A55.bp.flashloader1**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A55.bp.generic_watchdog**

ARM Generic Watchdog.
Type: `MemoryMappedGenericWatchdog`.

**FVP_Base_Cortex_A55.bp.hdlcd0**

ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).
Type: `PL370_HDLCD`.

**FVP_Base_Cortex_A55.bp.hdlcd0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55.bp.hdlcd0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55.bp.hdlcd0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55.bp.hdlcd0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55.bp.hdlcd0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A55.bp.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_Base_Cortex_A55.bp.mmc**

Generic Multimedia Card.
Type: `MMC`.

**FVP_Base_Cortex_A55.bp.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55.bp.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55.bp.mmc.timer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55.bp.mmc.timer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55.bp.mpe**
Encrypt memory transactions for each encryption context with an independent key to prevent mismatch access.
Type: `PVMemoryProtectionEngine`.

**FVP_Base_Cortex_A55.bp.nontrustedrom**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A55.bp.nontrustedromloader**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A55.bp.ns_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55.bp.pl011_uart0**
ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A55.bp.pl011_uart0.clk_divider**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55.bp.pl011_uart0.timer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A55.bp.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55.bp.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55.bp.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55.bp.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A55.bp.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55.bp.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55.bp.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55.bp.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A55.bp.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55.bp.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A55.bp.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55.bp.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55.bp.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55.bp.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55.bp.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55.bp.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A55.bp.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55.bp.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55.bp.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55.bp.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55.bp.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55.bp.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

**FVP_Base_Cortex_A55.bp.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55.bp.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55.bp.pl031_rtc.timer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55.bp.pl031_rtc.timer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55.bp.pl041_aaci**
ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_Base_Cortex_A55.bp.pl041_aaci.timer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55.bp.pl041_aaci.timer.timer**
A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55.bp.pl041_aaci.timer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55.bp.pl041_aaci.timer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55.bp.pl050_kmi0**
ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A55.bp.pl050_kmi0.clk_divider**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55.bp.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A55.bp.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55.bp.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_Base_Cortex_A55.bp.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

**FVP_Base_Cortex_A55.bp.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55.bp.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55.bp.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55.bp.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A55.bp.pl111_clcd_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A55.bp.pl180_mci**

ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

**FVP_Base_Cortex_A55.bp.ps2keyboard**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

**FVP_Base_Cortex_A55.bp.ps2keyboard.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55.bp.ps2keyboard.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55.bp.ps2keyboard.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55.bp.ps2keyboard.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55.bp.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

**FVP_Base_Cortex_A55.bp.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55.bp.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55.bp.ps2mouse.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55.bp.ps2mouse.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55.bp.psram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55.bp.refcounter**

Memory Mapped Counter Module for Generic Timers.
Type: `MemoryMappedCounterModule`.

**FVP_Base_Cortex_A55.bp.reset_or**

Or Gate.
Type: `OrGate`.

**FVP_Base_Cortex_A55.bp.rl_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55.bp.rt_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55.bp.s_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55.bp.secureDRAM**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A55.bp.secureSRAM**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55.bp.secureflash**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A55.bp.secureflashloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A55.bp.smsc_91c111**

SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_Base_Cortex_A55.bp.sp805_wdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A55.bp.sp805_wdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55.bp.sp805_wdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55.bp.sp805_wdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55.bp.sp805_wdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55.bp.sp810_sysctrl**

Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A55.bp.sp810_sysctrl.clkdiv_clk0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55.bp.sp810_sysctrl.clkdiv_clk1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55.bp.sp810_sysctrl.clkdiv_clk2**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55.bp.sp810_sysctrl.clkdiv_clk3**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55.bp.sram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55.bp.terminal_0**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A55.bp.terminal_1**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A55.bp.terminal_2**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A55.bp.terminal_3**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A55.bp.trusted_key_storage**

Trusted Root-Key Storage unit.
Type: `RootKeyStorage`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A55.bp.trusted_nv_counter**

Trusted Non-Volatile Counter unit.
Type: `NonVolatileCounter`.

**FVP_Base_Cortex_A55.bp.trusted_rng**

Random Number Generator unit.
Type: `RandomNumberGenerator`.

**FVP_Base_Cortex_A55.bp.trusted_watchdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A55.bp.trusted_watchdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55.bp.trusted_watchdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55.bp.trusted_watchdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55.bp.trusted_watchdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55.bp.tzc_400**

TrustZone Address Space Controller.
Type: `TZC_400`.

**FVP_Base_Cortex_A55.bp.ve_sysregs**

Type: `VE_SysRegs`.

**FVP_Base_Cortex_A55.bp.vedcc**

Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A55.bp.virtio_net**

VirtioNet device over MMIO transport.
Type: `VirtioNetMMIO`.

**FVP_Base_Cortex_A55.bp.virtio_net_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A55.bp.virtio_rng**

VirtioEntropy device over MMIO transport.
Type: `VirtioEntropyMMIO`.

**FVP_Base_Cortex_A55.bp.virtioblockdevice**

VirtioBlock device over MMIO transport.
Type: `VirtioBlockMMIO`.

**FVP_Base_Cortex_A55.bp.virtioblockdevice_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A55.bp.virtiop9device**

virtio P9 server.
Type: `VirtioP9Device`.

**FVP_Base_Cortex_A55.bp.virtiop9device_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A55.bp.vis**

Display window for VE using Visualisation library.
Type: `VEVisualisation`.

**FVP_Base_Cortex_A55.bp.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_Base_Cortex_A55.bp.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55.bp.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A55.bp.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55.bp.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55.bp.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55.bp.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55.bp.vram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55.cci400**

Cache Coherent Interconnect for AXI4 ACE.
Type: `CCI400`.

**FVP_Base_Cortex_A55.clockdivider0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55.cluster0**

ARM Cortex-A55 Cluster CT model.
Type: `Cluster_ARM_Cortex-A55`.

**FVP_Base_Cortex_A55.cluster0.DSU.shared_cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55.cluster0.cpu0**

ARM Cortex-A55 CT model.
Type: `ARM_Cortex-A55`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A55.cluster0.cpu0.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A55.cluster0.cpu0.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55.cluster0.cpu0.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55.cluster0.cpu0.l2cache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55.cluster0.cpu1**
ARM Cortex-A55 CT model.
Type: `ARM_Cortex-A55`.

**FVP_Base_Cortex_A55.cluster0.cpu1.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A55.cluster0.cpu1.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55.cluster0.cpu1.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55.cluster0.cpu1.l2cache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55.cluster0.cpu2**
ARM Cortex-A55 CT model.
Type: `ARM_Cortex-A55`.

**FVP_Base_Cortex_A55.cluster0.cpu2.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A55.cluster0.cpu2.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55.cluster0.cpu2.l1icache**
PV Cache.
Type: `PVCache`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A55.cluster0.cpu2.l2cache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55.cluster0.cpu3**
ARM Cortex-A55 CT model.
Type: `ARM_Cortex-A55`.

**FVP_Base_Cortex_A55.cluster0.cpu3.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A55.cluster0.cpu3.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55.cluster0.cpu3.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55.cluster0.cpu3.l2cache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55.cluster0.cpu4**
ARM Cortex-A55 CT model.
Type: `ARM_Cortex-A55`.

**FVP_Base_Cortex_A55.cluster0.cpu4.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A55.cluster0.cpu4.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55.cluster0.cpu4.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55.cluster0.cpu4.l2cache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55.cluster0.cpu5**
ARM Cortex-A55 CT model.
Type: `ARM_Cortex-A55`.

**FVP_Base_Cortex_A55.cluster0.cpu5.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A55.cluster0.cpu5.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55.cluster0.cpu5.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55.cluster0.cpu5.l2cache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55.cluster0.cpu6**
ARM Cortex-A55 CT model.
Type: `ARM_Cortex-A55`.

**FVP_Base_Cortex_A55.cluster0.cpu6.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A55.cluster0.cpu6.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55.cluster0.cpu6.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55.cluster0.cpu6.l2cache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55.cluster0.cpu7**
ARM Cortex-A55 CT model.
Type: `ARM_Cortex-A55`.

**FVP_Base_Cortex_A55.cluster0.cpu7.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A55.cluster0.cpu7.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55.cluster0.cpu7.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55.cluster0.cpu7.l2cache**
PV Cache.
Type: `PVCache`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A55.cluster0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A55.dapmemlogger**

Bus Logger.
Type: `PVBusLogger`.

**FVP_Base_Cortex_A55.elfloader**

ELF loader component.
Type: `ElfLoader`.

**FVP_Base_Cortex_A55.gic_distributor**

GIC Interrupt Redistribution Infrastucture component.
Type: `GIC_IRI`.

**FVP_Base_Cortex_A55.pctl**

Base Platforms Power Controller.
Type: `Base_PowerController`.

**FVP_Base_Cortex_A55.pctl.timer_reset**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55.pctl.timer_reset.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55.pctl.timer_reset.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55.pctl.timer_reset.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

# 13.20 FVP_Base_Cortex-A55+Cortex-A76

List of instances in FVP_Base_Cortex-A55+Cortex-A76.

## FVP_Base_Cortex-A55+Cortex-A76 instances

**FVP_Base_Cortex_A55_Cortex_A76**

Base Platform Compute Subsystem for ARMCortexA55CT_CortexA76CT.
Type: `FVP_Base_Cortex_A55_Cortex_A76`.

**FVP_Base_Cortex_A55_Cortex_A76.bp**

Peripherals and address map for the Base Platform.
Type: `BasePlatformPeripherals`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.Timer_0_1**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.Timer_0_1.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.Timer_0_1.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.Timer_0_1.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.Timer_0_1.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.Timer_2_3**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.Timer_2_3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A55_Cortex_A76.bp.Timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.Timer_2_3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.Timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.ap_refclk**

ARM Generic Timer.
Type: `MemoryMappedGenericTimer`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.clock300MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.clock32KHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A55_Cortex_A76.bp.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.dmc**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.dmc_phy**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.dummy_local_dap_rom**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.dummy_ram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.dummy_usb**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.exclusive_monitor**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A55_Cortex_A76.bp.flash0**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.flash1**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.flashloader0**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.flashloader1**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.generic_watchdog**

ARM Generic Watchdog.
Type: `MemoryMappedGenericWatchdog`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.hdlcd0**

ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).
Type: `PL370_HDLCD`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.hdlcd0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.hdlcd0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.hdlcd0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.hdlcd0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A55_Cortex_A76.bp.hdlcd0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.mmc**

Generic Multimedia Card.
Type: `MMC`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.mpe**

Encrypt memory transactions for each encryption context with an independent key to prevent mismatch access.
Type: `PVMemoryProtectionEngine`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.nontrustedrom**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.nontrustedromloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A55_Cortex_A76.bp.ns_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A55_Cortex_A76.bp.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A55_Cortex_A76.bp.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.pl111_clcd_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.pl180_mci**

ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.ps2keyboard**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.ps2keyboard.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A55_Cortex_A76.bp.ps2keyboard.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.ps2keyboard.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.ps2keyboard.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.ps2mouse.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.ps2mouse.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A55_Cortex_A76.bp.psram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.refcounter**
Memory Mapped Counter Module for Generic Timers.
Type: `MemoryMappedCounterModule`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.reset_or**
Or Gate.
Type: `OrGate`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.rl_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.rt_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.s_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.secureDRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.secureSRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.secureflash**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.secureflashloader**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.smsc_91c111**
SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.sp805_wdog**
ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.sp805_wdog.clocktimer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.sp805_wdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.sp805_wdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.sp805_wdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.sp810_sysctrl**

Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.sp810_sysctrl.clkdiv_clk0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.sp810_sysctrl.clkdiv_clk1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.sp810_sysctrl.clkdiv_clk2**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.sp810_sysctrl.clkdiv_clk3**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.sram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.terminal_0**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.terminal_1**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.terminal_2**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.terminal_3**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.trusted_key_storage**
Trusted Root-Key Storage unit.
Type: `RootKeyStorage`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.trusted_nv_counter**
Trusted Non-Volatile Counter unit.
Type: `NonVolatileCounter`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.trusted_rng**
Random Number Generator unit.
Type: `RandomNumberGenerator`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.trusted_watchdog**
ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.trusted_watchdog.clocktimer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.trusted_watchdog.clocktimer.timer**
A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.trusted_watchdog.clocktimer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.trusted_watchdog.clocktimer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.tzc_400**
TrustZone Address Space Controller.
Type: `TZC_400`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.ve_sysregs**
Type: `VE_SysRegs`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.vedcc**
Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.virtio_net**
VirtioNet device over MMIO transport.
Type: `VirtioNetMMIO`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.virtio_net_labeller**
Type: `Labeller`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.virtio_rng**
VirtioEntropy device over MMIO transport.
Type: `VirtioEntropyMMIO`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.virtioblockdevice**
VirtioBlock device over MMIO transport.
Type: `VirtioBlockMMIO`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.virtioblockdevice_labeller**
Type: `Labeller`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.virtiop9device**
virtio P9 server.
Type: `VirtioP9Device`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.virtiop9device_labeller**
Type: `Labeller`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.vis**
Display window for VE using Visualisation library.
Type: `VEVisualisation`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A55_Cortex_A76.bp.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55_Cortex_A76.bp.vram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A55_Cortex_A76.cci400**

Cache Coherent Interconnect for AXI4 ACE.
Type: `CCI400`.

**FVP_Base_Cortex_A55_Cortex_A76.clockdivider0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55_Cortex_A76.cluster0**

ARM Cortex-A55_Cortex-A76 Cluster CT model.
Type: `Cluster_ARM_Cortex-A55_Cortex-A76`.

**FVP_Base_Cortex_A55_Cortex_A76.cluster0.DSU.shared_cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55_Cortex_A76.cluster0.subcluster0**

ARM Cortex-A55 Cluster CT model.
Type: `Subcluster_ARM_Cortex-A55`.

**FVP_Base_Cortex_A55_Cortex_A76.cluster0.subcluster0.cpu0**

ARM Cortex-A55 CT model.
Type: `ARM_Cortex-A55`.

**FVP_Base_Cortex_A55_Cortex_A76.cluster0.subcluster0.cpu0.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55_Cortex_A76.cluster0.subcluster0.cpu0.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55_Cortex_A76.cluster0.subcluster0.cpu0.l2cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55_Cortex_A76.cluster0.subcluster1**

ARM Cortex-A76 Cluster CT model.
Type: `Subcluster_ARM_Cortex-A76`.

**FVP_Base_Cortex_A55_Cortex_A76.cluster0.subcluster1.cpu0**

ARM Cortex-A76 CT model.
Type: `ARM_Cortex-A76`.

**FVP_Base_Cortex_A55_Cortex_A76.cluster0.subcluster1.cpu0.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55_Cortex_A76.cluster0.subcluster1.cpu0.l1icache**

PV Cache.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `PVCache`.

**FVP_Base_Cortex_A55_Cortex_A76.cluster0.subcluster1.cpu0.l2cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55_Cortex_A76.cluster0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A55_Cortex_A76.dapmemlogger**

Bus Logger.
Type: `PVBusLogger`.

**FVP_Base_Cortex_A55_Cortex_A76.elfloader**

ELF loader component.
Type: `ElfLoader`.

**FVP_Base_Cortex_A55_Cortex_A76.gic_distributor**

GIC Interrupt Redistribution Infrastucture component.
Type: `GIC_IRI`.

**FVP_Base_Cortex_A55_Cortex_A76.pctl**

Base Platforms Power Controller.
Type: `Base_PowerController`.

**FVP_Base_Cortex_A55_Cortex_A76.pctl.timer_reset**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55_Cortex_A76.pctl.timer_reset.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55_Cortex_A76.pctl.timer_reset.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55_Cortex_A76.pctl.timer_reset.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

# 13.21 FVP_Base_Cortex-A55x1+Cortex-A75x1

List of instances in FVP_Base_Cortex-A55x1+Cortex-A75x1.

## FVP_Base_Cortex-A55x1+Cortex-A75x1 instances

**FVP_Base_Cortex_A55x1_Cortex_A75x1**

Base Platform Compute Subsystem for ARMCortexA55x1CT_CortexA75x1CT.
Type: `FVP_Base_Cortex_A55x1_Cortex_A75x1`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp**

Peripherals and address map for the Base Platform.
Type: `BasePlatformPeripherals`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.Timer_0_1**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.Timer_0_1.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.Timer_0_1.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.Timer_0_1.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.Timer_0_1.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.Timer_2_3**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.Timer_2_3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.Timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.Timer_2_3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.Timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.ap_refclk**

ARM Generic Timer.
Type: `MemoryMappedGenericTimer`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.clock300MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.clock32KHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.dmc**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.dmc_phy**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.dummy_local_dap_rom**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.dummy_ram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.dummy_usb**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.exclusive_monitor**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.flash0**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.flash1**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.flashloader0**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.flashloader1**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.generic_watchdog**

ARM Generic Watchdog.
Type: `MemoryMappedGenericWatchdog`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.hdlcd0**

ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).
Type: `PL370_HDLCD`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.hdlcd0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.hdlcd0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.hdlcd0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.hdlcd0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.hdlcd0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.mmc**

Generic Multimedia Card.
Type: `MMC`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.mpe**

Encrypt memory transactions for each encryption context with an independent key to prevent mismatch access.
Type: `PVMemoryProtectionEngine`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.nontrustedrom**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.nontrustedromloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.ns_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.pl111_clcd_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.pl180_mci**

ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.ps2keyboard**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.ps2keyboard.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.ps2keyboard.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.ps2keyboard.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.ps2keyboard.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.ps2mouse.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.ps2mouse.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.psram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.refcounter**
Memory Mapped Counter Module for Generic Timers.
Type: `MemoryMappedCounterModule`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.reset_or**
Or Gate.
Type: `OrGate`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.rl_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.rt_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.s_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.secureDRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.secureSRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.secureflash**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.secureflashloader**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.smsc_91c111**
SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.sp805_wdog**
ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.sp805_wdog.clocktimer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.sp805_wdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.sp805_wdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.sp805_wdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.sp810_sysctrl**

Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.sp810_sysctrl.clkdiv_clk0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.sp810_sysctrl.clkdiv_clk1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.sp810_sysctrl.clkdiv_clk2**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.sp810_sysctrl.clkdiv_clk3**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.sram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.terminal_0**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.terminal_1**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.terminal_2**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.terminal_3**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.trusted_key_storage**
Trusted Root-Key Storage unit.
Type: `RootKeyStorage`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.trusted_nv_counter**
Trusted Non-Volatile Counter unit.
Type: `NonVolatileCounter`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.trusted_rng**
Random Number Generator unit.
Type: `RandomNumberGenerator`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.trusted_watchdog**
ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.trusted_watchdog.clocktimer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.trusted_watchdog.clocktimer.timer**
A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.trusted_watchdog.clocktimer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.trusted_watchdog.clocktimer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.tzc_400**
TrustZone Address Space Controller.
Type: `TZC_400`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.ve_sysregs**
Type: `VE_SysRegs`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.vedcc**
Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.virtio_net**
VirtioNet device over MMIO transport.
Type: `VirtioNetMMIO`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.virtio_net_labeller**
Type: `Labeller`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.virtio_rng**
VirtioEntropy device over MMIO transport.
Type: `VirtioEntropyMMIO`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.virtioblockdevice**
VirtioBlock device over MMIO transport.
Type: `VirtioBlockMMIO`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.virtioblockdevice_labeller**
Type: `Labeller`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.virtiop9device**
virtio P9 server.
Type: `VirtioP9Device`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.virtiop9device_labeller**
Type: `Labeller`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.vis**
Display window for VE using Visualisation library.
Type: `VEVisualisation`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.bp.vram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A55x1_Cortex_A75x1.cci400**

Cache Coherent Interconnect for AXI4 ACE.
Type: `CCI400`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.clockdivider0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.cluster0**

ARM Cortex-A55_Cortex-A75 Cluster CT model.
Type: `Cluster_ARM_Cortex-A55_Cortex-A75`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.cluster0.DSU.shared_cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.cluster0.subcluster0**

ARM Cortex-A55 Cluster CT model.
Type: `Subcluster_ARM_Cortex-A55`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.cluster0.subcluster0.cpu0**

ARM Cortex-A55 CT model.
Type: `ARM_Cortex-A55`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.cluster0.subcluster0.cpu0.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.cluster0.subcluster0.cpu0.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.cluster0.subcluster0.cpu0.l2cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.cluster0.subcluster1**

ARM Cortex-A75 Cluster CT model.
Type: `Subcluster_ARM_Cortex-A75`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.cluster0.subcluster1.cpu0**

ARM Cortex-A75 CT model.
Type: `ARM_Cortex-A75`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.cluster0.subcluster1.cpu0.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.cluster0.subcluster1.cpu0.l1icache**

PV Cache.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `PVCache`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.cluster0.subcluster1.cpu0.l2cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.cluster0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.dapmemlogger**

Bus Logger.
Type: `PVBusLogger`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.elfloader**

ELF loader component.
Type: `ElfLoader`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.gic_distributor**

GIC Interrupt Redistribution Infrastucture component.
Type: `GIC_IRI`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.pctl**

Base Platforms Power Controller.
Type: `Base_PowerController`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.pctl.timer_reset**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.pctl.timer_reset.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.pctl.timer_reset.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55x1_Cortex_A75x1.pctl.timer_reset.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

# 13.22 FVP_Base_Cortex-A55x2+Cortex-A75x2

List of instances in FVP_Base_Cortex-A55x2+Cortex-A75x2.

## FVP_Base_Cortex-A55x2+Cortex-A75x2 instances

**FVP_Base_Cortex_A55x2_Cortex_A75x2**

Base Platform Compute Subsystem for ARMCortexA55x2CT_CortexA75x2CT.
Type: `FVP_Base_Cortex_A55x2_Cortex_A75x2`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp**

Peripherals and address map for the Base Platform.
Type: `BasePlatformPeripherals`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.Timer_0_1**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.Timer_0_1.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.Timer_0_1.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.Timer_0_1.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.Timer_0_1.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.Timer_2_3**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.Timer_2_3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.Timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.Timer_2_3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.Timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.ap_refclk**

ARM Generic Timer.
Type: `MemoryMappedGenericTimer`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.clock300MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.clock32KHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.dmc**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.dmc_phy**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.dummy_local_dap_rom**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.dummy_ram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.dummy_usb**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.exclusive_monitor**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.flash0**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.flash1**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.flashloader0**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.flashloader1**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.generic_watchdog**

ARM Generic Watchdog.
Type: `MemoryMappedGenericWatchdog`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.hdlcd0**

ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).
Type: `PL370_HDLCD`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.hdlcd0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.hdlcd0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.hdlcd0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.hdlcd0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.hdlcd0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.mmc**

Generic Multimedia Card.
Type: `MMC`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.mpe**

Encrypt memory transactions for each encryption context with an independent key to prevent mismatch access.
Type: `PVMemoryProtectionEngine`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.nontrustedrom**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.nontrustedromloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.ns_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.pl111_clcd_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.pl180_mci**

ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.ps2keyboard**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.ps2keyboard.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.ps2keyboard.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.ps2keyboard.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.ps2keyboard.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.ps2mouse.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.ps2mouse.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.psram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.refcounter**
Memory Mapped Counter Module for Generic Timers.
Type: `MemoryMappedCounterModule`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.reset_or**
Or Gate.
Type: `OrGate`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.rl_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.rt_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.s_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.secureDRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.secureSRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.secureflash**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.secureflashloader**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.smsc_91c111**
SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.sp805_wdog**
ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.sp805_wdog.clocktimer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.sp805_wdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.sp805_wdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.sp805_wdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.sp810_sysctrl**

Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.sp810_sysctrl.clkdiv_clk0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.sp810_sysctrl.clkdiv_clk1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.sp810_sysctrl.clkdiv_clk2**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.sp810_sysctrl.clkdiv_clk3**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.sram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.terminal_0**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.terminal_1**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.terminal_2**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.terminal_3**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.trusted_key_storage**
Trusted Root-Key Storage unit.
Type: `RootKeyStorage`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.trusted_nv_counter**
Trusted Non-Volatile Counter unit.
Type: `NonVolatileCounter`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.trusted_rng**
Random Number Generator unit.
Type: `RandomNumberGenerator`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.trusted_watchdog**
ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.trusted_watchdog.clocktimer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.trusted_watchdog.clocktimer.timer**
A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.trusted_watchdog.clocktimer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.trusted_watchdog.clocktimer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.tzc_400**
TrustZone Address Space Controller.
Type: `TZC_400`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.ve_sysregs**
Type: `VE_SysRegs`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.vedcc**
Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.virtio_net**
VirtioNet device over MMIO transport.
Type: `VirtioNetMMIO`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.virtio_net_labeller**
Type: `Labeller`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.virtio_rng**
VirtioEntropy device over MMIO transport.
Type: `VirtioEntropyMMIO`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.virtioblockdevice**
VirtioBlock device over MMIO transport.
Type: `VirtioBlockMMIO`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.virtioblockdevice_labeller**
Type: `Labeller`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.virtiop9device**
virtio P9 server.
Type: `VirtioP9Device`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.virtiop9device_labeller**
Type: `Labeller`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.vis**
Display window for VE using Visualisation library.
Type: `VEVisualisation`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.bp.vram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A55x2_Cortex_A75x2.cci400**

Cache Coherent Interconnect for AXI4 ACE.
Type: `CCI400`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.clockdivider0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.cluster0**

ARM Cortex-A55_Cortex-A75 Cluster CT model.
Type: `Cluster_ARM_Cortex-A55_Cortex-A75`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.cluster0.DSU.shared_cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.cluster0.subcluster0**

ARM Cortex-A55 Cluster CT model.
Type: `Subcluster_ARM_Cortex-A55`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.cluster0.subcluster0.cpu0**

ARM Cortex-A55 CT model.
Type: `ARM_Cortex-A55`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.cluster0.subcluster0.cpu0.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.cluster0.subcluster0.cpu0.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.cluster0.subcluster0.cpu0.l2cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.cluster0.subcluster0.cpu1**

ARM Cortex-A55 CT model.
Type: `ARM_Cortex-A55`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.cluster0.subcluster0.cpu1.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.cluster0.subcluster0.cpu1.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.cluster0.subcluster0.cpu1.l2cache**

PV Cache.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `PVCache`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.cluster0.subcluster1**

ARM Cortex-A75 Cluster CT model.
Type: `Subcluster_ARM_Cortex-A75`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.cluster0.subcluster1.cpu0**

ARM Cortex-A75 CT model.
Type: `ARM_Cortex-A75`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.cluster0.subcluster1.cpu0.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.cluster0.subcluster1.cpu0.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.cluster0.subcluster1.cpu0.l2cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.cluster0.subcluster1.cpu1**

ARM Cortex-A75 CT model.
Type: `ARM_Cortex-A75`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.cluster0.subcluster1.cpu1.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.cluster0.subcluster1.cpu1.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.cluster0.subcluster1.cpu1.l2cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.cluster0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.dapmemlogger**

Bus Logger.
Type: `PVBusLogger`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.elfloader**

ELF loader component.
Type: `ElfLoader`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.gic_distributor**

GIC Interrupt Redistribution Infrastucture component.
Type: `GIC_IRI`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A55x2_Cortex_A75x2.pctl**

Base Platforms Power Controller.
Type: `Base_PowerController`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.pctl.timer_reset**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.pctl.timer_reset.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.pctl.timer_reset.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55x2_Cortex_A75x2.pctl.timer_reset.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

## 13.23 FVP_Base_Cortex-A55x4+Cortex-A75x1

List of instances in FVP_Base_Cortex-A55x4+Cortex-A75x1.

### FVP_Base_Cortex-A55x4+Cortex-A75x1 instances

**FVP_Base_Cortex_A55x4_Cortex_A75x1**

Base Platform Compute Subsystem for ARMCortexA55x4CT_CortexA75x1CT.
Type: `FVP_Base_Cortex_A55x4_Cortex_A75x1`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp**

Peripherals and address map for the Base Platform.
Type: `BasePlatformPeripherals`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.Timer_0_1**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.Timer_0_1.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.Timer_0_1.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.Timer_0_1.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.Timer_0_1.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.Timer_2_3**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.Timer_2_3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.Timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.Timer_2_3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.Timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.ap_refclk**

ARM Generic Timer.
Type: `MemoryMappedGenericTimer`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.clock300MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.clock32KHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ClockDivider`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.dmc**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.dmc_phy**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.dummy_local_dap_rom**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.dummy_ram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.dummy_usb**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.exclusive_monitor**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.flash0**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.flash1**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.flashloader0**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.flashloader1**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.generic_watchdog**

ARM Generic Watchdog.
Type: `MemoryMappedGenericWatchdog`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.hdlcd0**

ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).
Type: `PL370_HDLCD`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.hdlcd0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.hdlcd0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.hdlcd0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.hdlcd0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.hdlcd0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.mmc**

Generic Multimedia Card.
Type: `MMC`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.mpe**

Encrypt memory transactions for each encryption context with an independent key to prevent mismatch access.
Type: `PVMemoryProtectionEngine`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.nontrustedrom**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.nontrustedromloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.ns_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.pl011_uart1.timer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.pl011_uart1.timer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.pl011_uart2**
ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.pl011_uart2.clk_divider**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.pl011_uart2.timer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.pl011_uart2.timer.timer**
A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.pl011_uart2.timer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.pl011_uart2.timer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `SchedulerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.pl111_clcd_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.pl180_mci**

ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.ps2keyboard**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.ps2keyboard.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.ps2keyboard.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.ps2keyboard.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.ps2keyboard.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `PS2Mouse`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.ps2mouse.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.ps2mouse.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.psram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.refcounter**

Memory Mapped Counter Module for Generic Timers.
Type: `MemoryMappedCounterModule`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.reset_or**

Or Gate.
Type: `OrGate`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.rl_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.rt_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.s_dram**

RAM device, can be dynamic or static ram.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `RAMDevice`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.secureDRAM**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.secureSRAM**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.secureflash**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.secureflashloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.smsc_91c111**

SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.sp805_wdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.sp805_wdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.sp805_wdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.sp805_wdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.sp805_wdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.sp810_sysctrl**

Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.sp810_sysctrl.clkdiv_clk0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.sp810_sysctrl.clkdiv_clk1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.sp810_sysctrl.clkdiv_clk2**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.sp810_sysctrl.clkdiv_clk3**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.sram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.terminal_0**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.terminal_1**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.terminal_2**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.terminal_3**

Telnet terminal interface.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `TelnetTerminal`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.trusted_key_storage**

Trusted Root-Key Storage unit.
Type: `RootKeyStorage`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.trusted_nv_counter**

Trusted Non-Volatile Counter unit.
Type: `NonVolatileCounter`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.trusted_rng**

Random Number Generator unit.
Type: `RandomNumberGenerator`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.trusted_watchdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.trusted_watchdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.trusted_watchdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.trusted_watchdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.trusted_watchdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.tzc_400**

TrustZone Address Space Controller.
Type: `TZC_400`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.ve_sysregs**

Type: `VE_SysRegs`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.vedcc**

Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.virtio_net**

VirtioNet device over MMIO transport.
Type: `VirtioNetMMIO`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.virtio_net_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.virtio_rng**

VirtioEntropy device over MMIO transport.
Type: `VirtioEntropyMMIO`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.virtioblockdevice**

VirtioBlock device over MMIO transport.
Type: `VirtioBlockMMIO`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.virtioblockdevice_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.virtiop9device**

virtio P9 server.
Type: `VirtioP9Device`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.virtiop9device_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.vis**

Display window for VE using Visualisation library.
Type: `VEVisualisation`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.bp.vram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.cci400**

Cache Coherent Interconnect for AXI4 ACE.
Type: `CCI400`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.clockdivider0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.cluster0**

ARM Cortex-A55_Cortex-A75 Cluster CT model.
Type: `Cluster_ARM_Cortex-A55_Cortex-A75`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.cluster0.DSU.shared_cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.cluster0.subcluster0**

ARM Cortex-A55 Cluster CT model.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `Subcluster_ARM_Cortex-A55`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.cluster0.subcluster0.cpu0**

ARM Cortex-A55 CT model.
Type: `ARM_Cortex-A55`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.cluster0.subcluster0.cpu0.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.cluster0.subcluster0.cpu0.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.cluster0.subcluster0.cpu0.l2cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.cluster0.subcluster0.cpu1**

ARM Cortex-A55 CT model.
Type: `ARM_Cortex-A55`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.cluster0.subcluster0.cpu1.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.cluster0.subcluster0.cpu1.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.cluster0.subcluster0.cpu1.l2cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.cluster0.subcluster0.cpu2**

ARM Cortex-A55 CT model.
Type: `ARM_Cortex-A55`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.cluster0.subcluster0.cpu2.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.cluster0.subcluster0.cpu2.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.cluster0.subcluster0.cpu2.l2cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.cluster0.subcluster0.cpu3**

ARM Cortex-A55 CT model.
Type: `ARM_Cortex-A55`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A55x4_Cortex_A75x1.cluster0.subcluster0.cpu3.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.cluster0.subcluster0.cpu3.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.cluster0.subcluster0.cpu3.l2cache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.cluster0.subcluster1**
ARM Cortex-A75 Cluster CT model.
Type: `Subcluster_ARM_Cortex-A75`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.cluster0.subcluster1.cpu0**
ARM Cortex-A75 CT model.
Type: `ARM_Cortex-A75`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.cluster0.subcluster1.cpu0.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.cluster0.subcluster1.cpu0.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.cluster0.subcluster1.cpu0.l2cache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.cluster0_labeller**
Type: `Labeller`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.dapmemlogger**
Bus Logger.
Type: `PVBusLogger`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.elfloader**
ELF loader component.
Type: `ElfLoader`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.gic_distributor**
GIC Interrupt Redistribution Infrastucture component.
Type: `GIC_IRI`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.pctl**
Base Platforms Power Controller.
Type: `Base_PowerController`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A55x4_Cortex_A75x1.pctl.timer_reset**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.pctl.timer_reset.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.pctl.timer_reset.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A75x1.pctl.timer_reset.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

## 13.24 FVP_Base_Cortex-A55x4+Cortex-A75x2

List of instances in FVP_Base_Cortex-A55x4+Cortex-A75x2.

### FVP_Base_Cortex-A55x4+Cortex-A75x2 instances

**FVP_Base_Cortex_A55x4_Cortex_A75x2**

Base Platform Compute Subsystem for ARMCortexA55x4CT_CortexA75x2CT.
Type: `FVP_Base_Cortex_A55x4_Cortex_A75x2`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp**

Peripherals and address map for the Base Platform.
Type: `BasePlatformPeripherals`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.Timer_0_1**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.Timer_0_1.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.Timer_0_1.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.Timer_0_1.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.Timer_0_1.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.Timer_2_3**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.Timer_2_3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.Timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.Timer_2_3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.Timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.ap_refclk**

ARM Generic Timer.
Type: `MemoryMappedGenericTimer`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.clock300MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.clock32KHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.dmc**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.dmc_phy**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.dummy_local_dap_rom**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.dummy_ram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.dummy_usb**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.exclusive_monitor**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.flash0**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.flash1**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.flashloader0**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.flashloader1**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.generic_watchdog**

ARM Generic Watchdog.
Type: `MemoryMappedGenericWatchdog`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.hdlcd0**

ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `PL370_HDLCD`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.hdlcd0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.hdlcd0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.hdlcd0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.hdlcd0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.hdlcd0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.mmc**

Generic Multimedia Card.
Type: `MMC`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.mpe**

Encrypt memory transactions for each encryption context with an independent key to prevent mismatch access.
Type: `PVMemoryProtectionEngine`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.nontrustedrom**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.nontrustedromloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.ns_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `SchedulerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.pl111_clcd_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.pl180_mci**

ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.ps2keyboard**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.ps2keyboard.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.ps2keyboard.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.ps2keyboard.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.ps2keyboard.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `PS2Mouse`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.ps2mouse.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.ps2mouse.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.psram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.refcounter**

Memory Mapped Counter Module for Generic Timers.
Type: `MemoryMappedCounterModule`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.reset_or**

Or Gate.
Type: `OrGate`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.rl_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.rt_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.s_dram**

RAM device, can be dynamic or static ram.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `RAMDevice`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.secureDRAM**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.secureSRAM**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.secureflash**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.secureflashloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.smsc_91c111**

SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.sp805_wdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.sp805_wdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.sp805_wdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.sp805_wdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.sp805_wdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.sp810_sysctrl**

Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.sp810_sysctrl.clkdiv_clk0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.sp810_sysctrl.clkdiv_clk1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.sp810_sysctrl.clkdiv_clk2**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.sp810_sysctrl.clkdiv_clk3**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.sram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.terminal_0**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.terminal_1**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.terminal_2**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.terminal_3**

Telnet terminal interface.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `TelnetTerminal`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.trusted_key_storage**

Trusted Root-Key Storage unit.
Type: `RootKeyStorage`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.trusted_nv_counter**

Trusted Non-Volatile Counter unit.
Type: `NonVolatileCounter`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.trusted_rng**

Random Number Generator unit.
Type: `RandomNumberGenerator`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.trusted_watchdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.trusted_watchdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.trusted_watchdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.trusted_watchdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.trusted_watchdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.tzc_400**

TrustZone Address Space Controller.
Type: `TZC_400`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.ve_sysregs**

Type: `VE_SysRegs`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.vedcc**

Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.virtio_net**

VirtioNet device over MMIO transport.
Type: `VirtioNetMMIO`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.virtio_net_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.virtio_rng**

VirtioEntropy device over MMIO transport.
Type: `VirtioEntropyMMIO`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.virtioblockdevice**

VirtioBlock device over MMIO transport.
Type: `VirtioBlockMMIO`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.virtioblockdevice_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.virtiop9device**

virtio P9 server.
Type: `VirtioP9Device`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.virtiop9device_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.vis**

Display window for VE using Visualisation library.
Type: `VEVisualisation`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.bp.vram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.cci400**

Cache Coherent Interconnect for AXI4 ACE.
Type: `CCI400`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.clockdivider0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.cluster0**

ARM Cortex-A55_Cortex-A75 Cluster CT model.
Type: `Cluster_ARM_Cortex-A55_Cortex-A75`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.cluster0.DSU.shared_cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.cluster0.subcluster0**

ARM Cortex-A55 Cluster CT model.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `Subcluster_ARM_Cortex-A55`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.cluster0.subcluster0.cpu0**

ARM Cortex-A55 CT model.
Type: `ARM_Cortex-A55`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.cluster0.subcluster0.cpu0.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.cluster0.subcluster0.cpu0.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.cluster0.subcluster0.cpu0.l2cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.cluster0.subcluster0.cpu1**

ARM Cortex-A55 CT model.
Type: `ARM_Cortex-A55`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.cluster0.subcluster0.cpu1.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.cluster0.subcluster0.cpu1.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.cluster0.subcluster0.cpu1.l2cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.cluster0.subcluster0.cpu2**

ARM Cortex-A55 CT model.
Type: `ARM_Cortex-A55`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.cluster0.subcluster0.cpu2.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.cluster0.subcluster0.cpu2.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.cluster0.subcluster0.cpu2.l2cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.cluster0.subcluster0.cpu3**

ARM Cortex-A55 CT model.
Type: `ARM_Cortex-A55`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A55x4_Cortex_A75x2.cluster0.subcluster0.cpu3.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.cluster0.subcluster0.cpu3.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.cluster0.subcluster0.cpu3.l2cache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.cluster0.subcluster1**
ARM Cortex-A75 Cluster CT model.
Type: `Subcluster_ARM_Cortex-A75`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.cluster0.subcluster1.cpu0**
ARM Cortex-A75 CT model.
Type: `ARM_Cortex-A75`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.cluster0.subcluster1.cpu0.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.cluster0.subcluster1.cpu0.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.cluster0.subcluster1.cpu0.l2cache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.cluster0.subcluster1.cpu1**
ARM Cortex-A75 CT model.
Type: `ARM_Cortex-A75`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.cluster0.subcluster1.cpu1.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.cluster0.subcluster1.cpu1.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.cluster0.subcluster1.cpu1.l2cache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.cluster0_labeller**
Type: `Labeller`.

**FVP_Base_Cortex_A55x4_Cortex_A75x2.dapmemlogger**
Bus Logger.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `PVBusLogger`.

**`FVP_Base_Cortex_A55x4_Cortex_A75x2.elfloader`**

ELF loader component.
Type: `ElfLoader`.

**`FVP_Base_Cortex_A55x4_Cortex_A75x2.gic_distributor`**

GIC Interrupt Redistribution Infrastucture component.
Type: `GIC_IRI`.

**`FVP_Base_Cortex_A55x4_Cortex_A75x2.pctl`**

Base Platforms Power Controller.
Type: `Base_PowerController`.

**`FVP_Base_Cortex_A55x4_Cortex_A75x2.pctl.timer_reset`**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**`FVP_Base_Cortex_A55x4_Cortex_A75x2.pctl.timer_reset.timer`**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**`FVP_Base_Cortex_A55x4_Cortex_A75x2.pctl.timer_reset.timer.thread`**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**`FVP_Base_Cortex_A55x4_Cortex_A75x2.pctl.timer_reset.timer.thread_event`**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

## 13.25 FVP_Base_Cortex-A55x4+Cortex-A75x4

List of instances in FVP_Base_Cortex-A55x4+Cortex-A75x4.

### FVP_Base_Cortex-A55x4+Cortex-A75x4 instances

**`FVP_Base_Cortex_A55x4_Cortex_A75x4`**

Base Platform Compute Subsystem for ARMCortexA55x4CT_CortexA75x4CT.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `FVP_Base_Cortex_A55x4_Cortex_A75x4`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp**

Peripherals and address map for the Base Platform.
Type: `BasePlatformPeripherals`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.Timer_0_1**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.Timer_0_1.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.Timer_0_1.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.Timer_0_1.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.Timer_0_1.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.Timer_2_3**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.Timer_2_3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.Timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.Timer_2_3.counter0**

Internal component used by SP804 Timer module.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `CounterModule`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.Timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.ap_refclk**

ARM Generic Timer.
Type: `MemoryMappedGenericTimer`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.clock300MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.clock32KHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.dmc**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.dmc_phy**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.dummy_local_dap_rom**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.dummy_ram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.dummy_usb**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.exclusive_monitor**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.flash0**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.flash1**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.flashloader0**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.flashloader1**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.generic_watchdog**

ARM Generic Watchdog.
Type: `MemoryMappedGenericWatchdog`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.hdlcd0**

ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).
Type: `PL370_HDLCD`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.hdlcd0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.hdlcd0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.hdlcd0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.hdlcd0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.hdlcd0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.mmc**

Generic Multimedia Card.
Type: `MMC`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.mpe**

Encrypt memory transactions for each encryption context with an independent key to prevent mismatch access.
Type: `PVMemoryProtectionEngine`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.nontrustedrom**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.nontrustedromloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.ns_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.pl111_clcd.pl11x_clcd.timer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.pl111_clcd.pl11x_clcd.timer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.pl111_clcd_labeller**
Type: `Labeller`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.pl180_mci**
ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.ps2keyboard**
Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.ps2keyboard.ps2_clocktimer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.ps2keyboard.ps2_clocktimer.timer**
A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.ps2keyboard.ps2_clocktimer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.ps2keyboard.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.ps2mouse.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.ps2mouse.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.psram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.refcounter**

Memory Mapped Counter Module for Generic Timers.
Type: `MemoryMappedCounterModule`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.reset_or**

Or Gate.
Type: `OrGate`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.rl_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.rt_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.s_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.secureDRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.secureSRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.secureflash**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.secureflashloader**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.smsc_91c111**
SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.sp805_wdog**
ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.sp805_wdog.clocktimer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.sp805_wdog.clocktimer.timer**
A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.sp805_wdog.clocktimer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.sp805_wdog.clocktimer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.sp810_sysctrl**
Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.sp810_sysctrl.clkdiv_clk0**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.sp810_sysctrl.clkdiv_clk1**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.sp810_sysctrl.clkdiv_clk2**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.sp810_sysctrl.clkdiv_clk3**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.sram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.terminal_0**
Telnet terminal interface.
Type: `TelnetTerminal`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.terminal_1**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.terminal_2**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.terminal_3**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.trusted_key_storage**

Trusted Root-Key Storage unit.
Type: `RootKeyStorage`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.trusted_nv_counter**

Trusted Non-Volatile Counter unit.
Type: `NonVolatileCounter`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.trusted_rng**

Random Number Generator unit.
Type: `RandomNumberGenerator`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.trusted_watchdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.trusted_watchdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.trusted_watchdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.trusted_watchdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.trusted_watchdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.tzc_400**

TrustZone Address Space Controller.
Type: `TZC_400`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.ve_sysregs**

Type: `VE_SysRegs`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.vedcc**

Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.virtio_net**

VirtioNet device over MMIO transport.
Type: `VirtioNetMMIO`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.virtio_net_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.virtio_rng**

VirtioEntropy device over MMIO transport.
Type: `VirtioEntropyMMIO`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.virtioblockdevice**

VirtioBlock device over MMIO transport.
Type: `VirtioBlockMMIO`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.virtioblockdevice_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.virtiop9device**

virtio P9 server.
Type: `VirtioP9Device`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.virtiop9device_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.vis**

Display window for VE using Visualisation library.
Type: `VEVisualisation`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.bp.vram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.cci400**

Cache Coherent Interconnect for AXI4 ACE.
Type: `CCI400`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.clockdivider0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**`FVP_Base_Cortex_A55x4_Cortex_A75x4.cluster0`**
ARM Cortex-A55_Cortex-A75 Cluster CT model.
Type: `Cluster_ARM_Cortex-A55_Cortex-A75`.

**`FVP_Base_Cortex_A55x4_Cortex_A75x4.cluster0.DSU.shared_cache`**
PV Cache.
Type: `PVCache`.

**`FVP_Base_Cortex_A55x4_Cortex_A75x4.cluster0.subcluster0`**
ARM Cortex-A55 Cluster CT model.
Type: `Subcluster_ARM_Cortex-A55`.

**`FVP_Base_Cortex_A55x4_Cortex_A75x4.cluster0.subcluster0.cpu0`**
ARM Cortex-A55 CT model.
Type: `ARM_Cortex-A55`.

**`FVP_Base_Cortex_A55x4_Cortex_A75x4.cluster0.subcluster0.cpu0.l1dcache`**
PV Cache.
Type: `PVCache`.

**`FVP_Base_Cortex_A55x4_Cortex_A75x4.cluster0.subcluster0.cpu0.l1icache`**
PV Cache.
Type: `PVCache`.

**`FVP_Base_Cortex_A55x4_Cortex_A75x4.cluster0.subcluster0.cpu0.l2cache`**
PV Cache.
Type: `PVCache`.

**`FVP_Base_Cortex_A55x4_Cortex_A75x4.cluster0.subcluster0.cpu1`**
ARM Cortex-A55 CT model.
Type: `ARM_Cortex-A55`.

**`FVP_Base_Cortex_A55x4_Cortex_A75x4.cluster0.subcluster0.cpu1.l1dcache`**
PV Cache.
Type: `PVCache`.

**`FVP_Base_Cortex_A55x4_Cortex_A75x4.cluster0.subcluster0.cpu1.l1icache`**
PV Cache.
Type: `PVCache`.

**`FVP_Base_Cortex_A55x4_Cortex_A75x4.cluster0.subcluster0.cpu1.l2cache`**
PV Cache.
Type: `PVCache`.

**`FVP_Base_Cortex_A55x4_Cortex_A75x4.cluster0.subcluster0.cpu2`**
ARM Cortex-A55 CT model.
Type: `ARM_Cortex-A55`.

**`FVP_Base_Cortex_A55x4_Cortex_A75x4.cluster0.subcluster0.cpu2.l1dcache`**
PV Cache.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `PVCache`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.cluster0.subcluster0.cpu2.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.cluster0.subcluster0.cpu2.l2cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.cluster0.subcluster0.cpu3**

ARM Cortex-A55 CT model.
Type: `ARM_Cortex-A55`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.cluster0.subcluster0.cpu3.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.cluster0.subcluster0.cpu3.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.cluster0.subcluster0.cpu3.l2cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.cluster0.subcluster1**

ARM Cortex-A75 Cluster CT model.
Type: `Subcluster_ARM_Cortex-A75`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.cluster0.subcluster1.cpu0**

ARM Cortex-A75 CT model.
Type: `ARM_Cortex-A75`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.cluster0.subcluster1.cpu0.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.cluster0.subcluster1.cpu0.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.cluster0.subcluster1.cpu0.l2cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.cluster0.subcluster1.cpu1**

ARM Cortex-A75 CT model.
Type: `ARM_Cortex-A75`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.cluster0.subcluster1.cpu1.l1dcache**

PV Cache.
Type: `PVCache`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A55x4_Cortex_A75x4.cluster0.subcluster1.cpu1.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.cluster0.subcluster1.cpu1.l2cache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.cluster0.subcluster1.cpu2**
ARM Cortex-A75 CT model.
Type: `ARM_Cortex-A75`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.cluster0.subcluster1.cpu2.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.cluster0.subcluster1.cpu2.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.cluster0.subcluster1.cpu2.l2cache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.cluster0.subcluster1.cpu3**
ARM Cortex-A75 CT model.
Type: `ARM_Cortex-A75`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.cluster0.subcluster1.cpu3.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.cluster0.subcluster1.cpu3.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.cluster0.subcluster1.cpu3.l2cache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.cluster0_labeller**
Type: `Labeller`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.dapmemlogger**
Bus Logger.
Type: `PVBusLogger`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.elfloader**
ELF loader component.
Type: `ElfLoader`.

**FVP_Base_Cortex_A55x4_Cortex_A75x4.gic_distributor**
GIC Interrupt Redistribution Infrastucture component.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `GIC_IRI`.

**`FVP_Base_Cortex_A55x4_Cortex_A75x4.pctl`**

Base Platforms Power Controller.
Type: `Base_PowerController`.

**`FVP_Base_Cortex_A55x4_Cortex_A75x4.pctl.timer_reset`**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**`FVP_Base_Cortex_A55x4_Cortex_A75x4.pctl.timer_reset.timer`**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**`FVP_Base_Cortex_A55x4_Cortex_A75x4.pctl.timer_reset.timer.thread`**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**`FVP_Base_Cortex_A55x4_Cortex_A75x4.pctl.timer_reset.timer.thread_event`**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

## 13.26 FVP_Base_Cortex-A55x4+Cortex-A76x2

List of instances in FVP_Base_Cortex-A55x4+Cortex-A76x2.

### FVP_Base_Cortex-A55x4+Cortex-A76x2 instances

**`FVP_Base_Cortex_A55x4_Cortex_A76x2`**

Base Platform Compute Subsystem for ARMCortexA55x4CT_CortexA76x2CT.
Type: `FVP_Base_Cortex_A55x4_Cortex_A76x2`.

**`FVP_Base_Cortex_A55x4_Cortex_A76x2.bp`**

Peripherals and address map for the Base Platform.
Type: `BasePlatformPeripherals`.

**`FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.Timer_0_1`**

ARM Dual-Timer Module(SP804).

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `SP804_Timer`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.Timer_0_1.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.Timer_0_1.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.Timer_0_1.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.Timer_0_1.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.Timer_2_3**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.Timer_2_3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.Timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.Timer_2_3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.Timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.ap_refclk**

ARM Generic Timer.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `MemoryMappedGenericTimer`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.clock300MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.clock32KHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.clockdivider**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.dmc**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.dmc_phy**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.dummy_local_dap_rom**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.dummy_ram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.dummy_usb**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.exclusive_monitor**
Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.flash0**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.flash1**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.flashloader0**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.flashloader1**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.generic_watchdog**
ARM Generic Watchdog.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `MemoryMappedGenericWatchdog`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.hdlcd0**

ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).
Type: `PL370_HDLCD`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.hdlcd0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.hdlcd0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.hdlcd0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.hdlcd0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.hdlcd0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.mmc**

Generic Multimedia Card.
Type: `MMC`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.mpe**

Encrypt memory transactions for each encryption context with an independent key to prevent mismatch access.
Type: `PVMemoryProtectionEngine`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.nontrustedrom**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.nontrustedromloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.ns_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal()

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.pl111_clcd.pl11x_clcd.timer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.pl111_clcd.pl11x_clcd.timer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.pl111_clcd_labeller**
Type: `Labeller`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.pl180_mci**
ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.ps2keyboard**
Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.ps2keyboard.ps2_clocktimer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.ps2keyboard.ps2_clocktimer.timer**
A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.ps2keyboard.ps2_clocktimer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.ps2keyboard.ps2_clocktimer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.ps2mouse.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.ps2mouse.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.psram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.refcounter**

Memory Mapped Counter Module for Generic Timers.
Type: `MemoryMappedCounterModule`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.reset_or**

Or Gate.
Type: `OrGate`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.rl_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.rt_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.s_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.secureDRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.secureSRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.secureflash**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.secureflashloader**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.smsc_91c111**
SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.sp805_wdog**
ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.sp805_wdog.clocktimer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.sp805_wdog.clocktimer.timer**
A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.sp805_wdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.sp805_wdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.sp810_sysctrl**

Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.sp810_sysctrl.clkdiv_clk0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.sp810_sysctrl.clkdiv_clk1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.sp810_sysctrl.clkdiv_clk2**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.sp810_sysctrl.clkdiv_clk3**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.sram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.terminal_0**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.terminal_1**

Telnet terminal interface.
Type: `TelnetTerminal`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.terminal_2**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.terminal_3**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.trusted_key_storage**

Trusted Root-Key Storage unit.
Type: `RootKeyStorage`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.trusted_nv_counter**

Trusted Non-Volatile Counter unit.
Type: `NonVolatileCounter`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.trusted_rng**

Random Number Generator unit.
Type: `RandomNumberGenerator`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.trusted_watchdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.trusted_watchdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.trusted_watchdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.trusted_watchdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.trusted_watchdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.tzc_400**

TrustZone Address Space Controller.
Type: `TZC_400`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.ve_sysregs**

Type: `VE_SysRegs`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.vedcc**

Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.virtio_net**

VirtioNet device over MMIO transport.
Type: `VirtioNetMMIO`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.virtio_net_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.virtio_rng**

VirtioEntropy device over MMIO transport.
Type: `VirtioEntropyMMIO`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.virtioblockdevice**

VirtioBlock device over MMIO transport.
Type: `VirtioBlockMMIO`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.virtioblockdevice_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.virtiop9device**

virtio P9 server.
Type: `VirtioP9Device`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.virtiop9device_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.vis**

Display window for VE using Visualisation library.
Type: `VEVisualisation`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.bp.vram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.cci400**

Cache Coherent Interconnect for AXI4 ACE.
Type: `CCI400`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.clockdivider0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A55x4_Cortex_A76x2.cluster0**

ARM Cortex-A55_Cortex-A76 Cluster CT model.
Type: `Cluster_ARM_Cortex-A55_Cortex-A76`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.cluster0.DSU.shared_cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.cluster0.subcluster0**

ARM Cortex-A55 Cluster CT model.
Type: `Subcluster_ARM_Cortex-A55`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.cluster0.subcluster0.cpu0**

ARM Cortex-A55 CT model.
Type: `ARM_Cortex-A55`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.cluster0.subcluster0.cpu0.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.cluster0.subcluster0.cpu0.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.cluster0.subcluster0.cpu0.l2cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.cluster0.subcluster0.cpu1**

ARM Cortex-A55 CT model.
Type: `ARM_Cortex-A55`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.cluster0.subcluster0.cpu1.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.cluster0.subcluster0.cpu1.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.cluster0.subcluster0.cpu1.l2cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.cluster0.subcluster0.cpu2**

ARM Cortex-A55 CT model.
Type: `ARM_Cortex-A55`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.cluster0.subcluster0.cpu2.l1dcache**

PV Cache.
Type: `PVCache`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A55x4_Cortex_A76x2.cluster0.subcluster0.cpu2.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.cluster0.subcluster0.cpu2.l2cache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.cluster0.subcluster0.cpu3**
ARM Cortex-A55 CT model.
Type: `ARM_Cortex-A55`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.cluster0.subcluster0.cpu3.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.cluster0.subcluster0.cpu3.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.cluster0.subcluster0.cpu3.l2cache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.cluster0.subcluster1**
ARM Cortex-A76 Cluster CT model.
Type: `Subcluster_ARM_Cortex-A76`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.cluster0.subcluster1.cpu0**
ARM Cortex-A76 CT model.
Type: `ARM_Cortex-A76`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.cluster0.subcluster1.cpu0.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.cluster0.subcluster1.cpu0.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.cluster0.subcluster1.cpu0.l2cache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.cluster0.subcluster1.cpu1**
ARM Cortex-A76 CT model.
Type: `ARM_Cortex-A76`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.cluster0.subcluster1.cpu1.l1dcache**
PV Cache.
Type: `PVCache`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A55x4_Cortex_A76x2.cluster0.subcluster1.cpu1.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.cluster0.subcluster1.cpu1.l2cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.cluster0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.dapmemlogger**

Bus Logger.
Type: `PVBusLogger`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.elfloader**

ELF loader component.
Type: `ElfLoader`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.gic_distributor**

GIC Interrupt Redistribution Infrastucture component.
Type: `GIC_IRI`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.pctl**

Base Platforms Power Controller.
Type: `Base_PowerController`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.pctl.timer_reset**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.pctl.timer_reset.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A55x4_Cortex_A76x2.pctl.timer_reset.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A55x4_Cortex_A76x2.pctl.timer_reset.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

## 13.27 FVP_Base_Cortex-A57x1

List of instances in FVP_Base_Cortex-A57x1.

### FVP_Base_Cortex-A57x1 instances

**FVP_Base_Cortex_A57x1**

Base Platform Compute Subsystem for ARMCortexA57x1CT.
Type: `FVP_Base_Cortex_A57x1`.

**FVP_Base_Cortex_A57x1.bp**

Peripherals and address map for the Base Platform.
Type: `BasePlatformPeripherals`.

**FVP_Base_Cortex_A57x1.bp.Timer_0_1**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Cortex_A57x1.bp.Timer_0_1.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x1.bp.Timer_0_1.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x1.bp.Timer_0_1.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A57x1.bp.Timer_0_1.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A57x1.bp.Timer_2_3**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x1.bp.Timer_2_3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x1.bp.Timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x1.bp.Timer_2_3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A57x1.bp.Timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A57x1.bp.ap_refclk**

ARM Generic Timer.
Type: `MemoryMappedGenericTimer`.

**FVP_Base_Cortex_A57x1.bp.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**FVP_Base_Cortex_A57x1.bp.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x1.bp.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x1.bp.clock300MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x1.bp.clock32KHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x1.bp.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x1.bp.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x1.bp.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x1.bp.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x1.bp.dmc**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x1.bp.dmc_phy**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x1.bp.dummy_local_dap_rom**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x1.bp.dummy_ram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x1.bp.dummy_usb**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x1.bp.exclusive_monitor**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_Base_Cortex_A57x1.bp.flash0**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A57x1.bp.flash1**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A57x1.bp.flashloader0**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A57x1.bp.flashloader1**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A57x1.bp.generic_watchdog**

ARM Generic Watchdog.
Type: `MemoryMappedGenericWatchdog`.

**FVP_Base_Cortex_A57x1.bp.hdlcd0**

ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).
Type: `PL370_HDLCD`.

**FVP_Base_Cortex_A57x1.bp.hdlcd0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x1.bp.hdlcd0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x1.bp.hdlcd0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x1.bp.hdlcd0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x1.bp.hdlcd0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A57x1.bp.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_Base_Cortex_A57x1.bp.mmc**

Generic Multimedia Card.
Type: `MMC`.

**FVP_Base_Cortex_A57x1.bp.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x1.bp.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x1.bp.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x1.bp.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x1.bp.mpe**

Encrypt memory transactions for each encryption context with an independent key to prevent mismatch access.
Type: `PVMemoryProtectionEngine`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x1.bp.nontrustedrom**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A57x1.bp.nontrustedromloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A57x1.bp.ns_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x1.bp.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A57x1.bp.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x1.bp.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x1.bp.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x1.bp.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x1.bp.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x1.bp.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A57x1.bp.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x1.bp.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x1.bp.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x1.bp.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x1.bp.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x1.bp.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A57x1.bp.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x1.bp.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x1.bp.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x1.bp.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x1.bp.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x1.bp.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A57x1.bp.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x1.bp.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x1.bp.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x1.bp.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x1.bp.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x1.bp.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

**FVP_Base_Cortex_A57x1.bp.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x1.bp.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x1.bp.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x1.bp.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x1.bp.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_Base_Cortex_A57x1.bp.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x1.bp.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x1.bp.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x1.bp.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x1.bp.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A57x1.bp.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x1.bp.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A57x1.bp.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x1.bp.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_Base_Cortex_A57x1.bp.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

**FVP_Base_Cortex_A57x1.bp.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x1.bp.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x1.bp.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x1.bp.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x1.bp.pl111_clcd_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A57x1.bp.pl180_mci**

ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

**FVP_Base_Cortex_A57x1.bp.ps2keyboard**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x1.bp.ps2keyboard.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x1.bp.ps2keyboard.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x1.bp.ps2keyboard.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x1.bp.ps2keyboard.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x1.bp.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

**FVP_Base_Cortex_A57x1.bp.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x1.bp.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x1.bp.ps2mouse.ps2_clocktimer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x1.bp.ps2mouse.ps2_clocktimer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x1.bp.psram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x1.bp.refcounter**
Memory Mapped Counter Module for Generic Timers.
Type: `MemoryMappedCounterModule`.

**FVP_Base_Cortex_A57x1.bp.reset_or**
Or Gate.
Type: `OrGate`.

**FVP_Base_Cortex_A57x1.bp.rl_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x1.bp.rt_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x1.bp.s_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x1.bp.secureDRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x1.bp.secureSRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x1.bp.secureflash**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A57x1.bp.secureflashloader**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x1.bp.smsc_91c111**

SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_Base_Cortex_A57x1.bp.sp805_wdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A57x1.bp.sp805_wdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x1.bp.sp805_wdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x1.bp.sp805_wdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x1.bp.sp805_wdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x1.bp.sp810_sysctrl**

Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_Base_Cortex_A57x1.bp.sp810_sysctrl.clkdiv_clk0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x1.bp.sp810_sysctrl.clkdiv_clk1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x1.bp.sp810_sysctrl.clkdiv_clk2**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x1.bp.sp810_sysctrl.clkdiv_clk3**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x1.bp.sram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x1.bp.terminal_0**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A57x1.bp.terminal_1**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A57x1.bp.terminal_2**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A57x1.bp.terminal_3**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A57x1.bp.trusted_key_storage**
Trusted Root-Key Storage unit.
Type: `RootKeyStorage`.

**FVP_Base_Cortex_A57x1.bp.trusted_nv_counter**
Trusted Non-Volatile Counter unit.
Type: `NonVolatileCounter`.

**FVP_Base_Cortex_A57x1.bp.trusted_rng**
Random Number Generator unit.
Type: `RandomNumberGenerator`.

**FVP_Base_Cortex_A57x1.bp.trusted_watchdog**
ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x1.bp.trusted_watchdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x1.bp.trusted_watchdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x1.bp.trusted_watchdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x1.bp.trusted_watchdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x1.bp.tzc_400**

TrustZone Address Space Controller.
Type: `TZC_400`.

**FVP_Base_Cortex_A57x1.bp.ve_sysregs**

Type: `VE_SysRegs`.

**FVP_Base_Cortex_A57x1.bp.vedcc**

Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

**FVP_Base_Cortex_A57x1.bp.virtio_net**

VirtioNet device over MMIO transport.
Type: `VirtioNetMMIO`.

**FVP_Base_Cortex_A57x1.bp.virtio_net_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A57x1.bp.virtio_rng**

VirtioEntropy device over MMIO transport.
Type: `VirtioEntropyMMIO`.

**FVP_Base_Cortex_A57x1.bp.virtioblockdevice**

VirtioBlock device over MMIO transport.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `VirtioBlockMMIO`.

**FVP_Base_Cortex_A57x1.bp.virtioblockdevice_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A57x1.bp.virtiop9device**

virtio P9 server.
Type: `VirtioP9Device`.

**FVP_Base_Cortex_A57x1.bp.virtiop9device_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A57x1.bp.vis**

Display window for VE using Visualisation library.
Type: `VEVisualisation`.

**FVP_Base_Cortex_A57x1.bp.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_Base_Cortex_A57x1.bp.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x1.bp.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x1.bp.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x1.bp.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x1.bp.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x1.bp.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x1.bp.vram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x1.cci400**

Cache Coherent Interconnect for AXI4 ACE.
Type: `CCI400`.

**FVP_Base_Cortex_A57x1.clockdivider0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x1.cluster0**

ARM Cortex-A57 Cluster CT model.
Type: `Cluster_ARM_Cortex-A57`.

**FVP_Base_Cortex_A57x1.cluster0.cpu0**

ARM Cortex-A57 CT model.
Type: `ARM_Cortex-A57`.

**FVP_Base_Cortex_A57x1.cluster0.cpu0.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A57x1.cluster0.cpu0.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A57x1.cluster0.cpu0.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A57x1.cluster0.l2_cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A57x1.cluster0_labeller**

Type: `Labeller`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x1.dapmemlogger**

Bus Logger.
Type: `PVBusLogger`.

**FVP_Base_Cortex_A57x1.elfloader**

ELF loader component.
Type: `ElfLoader`.

**FVP_Base_Cortex_A57x1.gic_distributor**

GIC Interrupt Redistribution Infrastucture component.
Type: `GIC_IRI`.

**FVP_Base_Cortex_A57x1.pctl**

Base Platforms Power Controller.
Type: `Base_PowerController`.

**FVP_Base_Cortex_A57x1.pctl.timer_reset**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x1.pctl.timer_reset.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x1.pctl.timer_reset.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x1.pctl.timer_reset.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

# 13.28 FVP_Base_Cortex-A57x1-A35x1

List of instances in FVP_Base_Cortex-A57x1-A35x1.

## FVP_Base_Cortex-A57x1-A35x1 instances

**FVP_Base_Cortex_A57x1_A35x1**

Base Platform Compute Subsystem for ARMCortexA57x1CT and ARMCortexA35x1CT.
Type: `FVP_Base_Cortex_A57x1_A35x1`.

**FVP_Base_Cortex_A57x1_A35x1.bp**

Peripherals and address map for the Base Platform.
Type: `BasePlatformPeripherals`.

**FVP_Base_Cortex_A57x1_A35x1.bp.Timer_0_1**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Cortex_A57x1_A35x1.bp.Timer_0_1.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x1_A35x1.bp.Timer_0_1.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x1_A35x1.bp.Timer_0_1.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A57x1_A35x1.bp.Timer_0_1.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A57x1_A35x1.bp.Timer_2_3**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Cortex_A57x1_A35x1.bp.Timer_2_3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x1_A35x1.bp.Timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x1_A35x1.bp.Timer_2_3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A57x1_A35x1.bp.Timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A57x1_A35x1.bp.ap_refclk**

ARM Generic Timer.
Type: `MemoryMappedGenericTimer`.

**FVP_Base_Cortex_A57x1_A35x1.bp.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**FVP_Base_Cortex_A57x1_A35x1.bp.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x1_A35x1.bp.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x1_A35x1.bp.clock300MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x1_A35x1.bp.clock32KHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x1_A35x1.bp.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x1_A35x1.bp.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x1_A35x1.bp.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x1_A35x1.bp.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x1_A35x1.bp.dmc**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x1_A35x1.bp.dmc_phy**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x1_A35x1.bp.dummy_local_dap_rom**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x1_A35x1.bp.dummy_ram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x1_A35x1.bp.dummy_usb**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x1_A35x1.bp.exclusive_monitor**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x1_A35x1.bp.flash0**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A57x1_A35x1.bp.flash1**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A57x1_A35x1.bp.flashloader0**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A57x1_A35x1.bp.flashloader1**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A57x1_A35x1.bp.generic_watchdog**

ARM Generic Watchdog.
Type: `MemoryMappedGenericWatchdog`.

**FVP_Base_Cortex_A57x1_A35x1.bp.hdlcd0**

ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).
Type: `PL370_HDLCD`.

**FVP_Base_Cortex_A57x1_A35x1.bp.hdlcd0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x1_A35x1.bp.hdlcd0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x1_A35x1.bp.hdlcd0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x1_A35x1.bp.hdlcd0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x1_A35x1.bp.hdlcd0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A57x1_A35x1.bp.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_Base_Cortex_A57x1_A35x1.bp.mmc**

Generic Multimedia Card.
Type: `MMC`.

**FVP_Base_Cortex_A57x1_A35x1.bp.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x1_A35x1.bp.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x1_A35x1.bp.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x1_A35x1.bp.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x1_A35x1.bp.mpe**

Encrypt memory transactions for each encryption context with an independent key to prevent mismatch access.
Type: `PVMemoryProtectionEngine`.

**FVP_Base_Cortex_A57x1_A35x1.bp.nontrustedrom**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A57x1_A35x1.bp.nontrustedromloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x1_A35x1.bp.ns_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x1_A35x1.bp.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A57x1_A35x1.bp.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x1_A35x1.bp.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x1_A35x1.bp.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x1_A35x1.bp.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x1_A35x1.bp.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x1_A35x1.bp.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A57x1_A35x1.bp.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x1_A35x1.bp.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x1_A35x1.bp.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x1_A35x1.bp.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x1_A35x1.bp.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x1_A35x1.bp.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A57x1_A35x1.bp.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x1_A35x1.bp.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x1_A35x1.bp.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x1_A35x1.bp.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x1_A35x1.bp.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x1_A35x1.bp.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A57x1_A35x1.bp.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x1_A35x1.bp.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x1_A35x1.bp.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x1_A35x1.bp.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x1_A35x1.bp.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x1_A35x1.bp.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

**FVP_Base_Cortex_A57x1_A35x1.bp.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x1_A35x1.bp.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x1_A35x1.bp.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x1_A35x1.bp.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x1_A35x1.bp.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_Base_Cortex_A57x1_A35x1.bp.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x1_A35x1.bp.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x1_A35x1.bp.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x1_A35x1.bp.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x1_A35x1.bp.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A57x1_A35x1.bp.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x1_A35x1.bp.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A57x1_A35x1.bp.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x1_A35x1.bp.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_Base_Cortex_A57x1_A35x1.bp.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x1_A35x1.bp.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x1_A35x1.bp.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x1_A35x1.bp.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x1_A35x1.bp.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x1_A35x1.bp.pl111_clcd_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A57x1_A35x1.bp.pl180_mci**

ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

**FVP_Base_Cortex_A57x1_A35x1.bp.ps2keyboard**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

**FVP_Base_Cortex_A57x1_A35x1.bp.ps2keyboard.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x1_A35x1.bp.ps2keyboard.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x1_A35x1.bp.ps2keyboard.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x1_A35x1.bp.ps2keyboard.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x1_A35x1.bp.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

**FVP_Base_Cortex_A57x1_A35x1.bp.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x1_A35x1.bp.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x1_A35x1.bp.ps2mouse.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x1_A35x1.bp.ps2mouse.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x1_A35x1.bp.psram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x1_A35x1.bp.refcounter**
Memory Mapped Counter Module for Generic Timers.
Type: `MemoryMappedCounterModule`.

**FVP_Base_Cortex_A57x1_A35x1.bp.reset_or**
Or Gate.
Type: `OrGate`.

**FVP_Base_Cortex_A57x1_A35x1.bp.rl_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x1_A35x1.bp.rt_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x1_A35x1.bp.s_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x1_A35x1.bp.secureDRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x1_A35x1.bp.secureSRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x1_A35x1.bp.secureflash**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A57x1_A35x1.bp.secureflashloader**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A57x1_A35x1.bp.smsc_91c111**
SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_Base_Cortex_A57x1_A35x1.bp.sp805_wdog**
ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A57x1_A35x1.bp.sp805_wdog.clocktimer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x1_A35x1.bp.sp805_wdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x1_A35x1.bp.sp805_wdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x1_A35x1.bp.sp805_wdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x1_A35x1.bp.sp810_sysctrl**

Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_Base_Cortex_A57x1_A35x1.bp.sp810_sysctrl.clkdiv_clk0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x1_A35x1.bp.sp810_sysctrl.clkdiv_clk1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x1_A35x1.bp.sp810_sysctrl.clkdiv_clk2**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x1_A35x1.bp.sp810_sysctrl.clkdiv_clk3**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x1_A35x1.bp.sram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x1_A35x1.bp.terminal_0**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A57x1_A35x1.bp.terminal_1**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A57x1_A35x1.bp.terminal_2**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A57x1_A35x1.bp.terminal_3**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A57x1_A35x1.bp.trusted_key_storage**
Trusted Root-Key Storage unit.
Type: `RootKeyStorage`.

**FVP_Base_Cortex_A57x1_A35x1.bp.trusted_nv_counter**
Trusted Non-Volatile Counter unit.
Type: `NonVolatileCounter`.

**FVP_Base_Cortex_A57x1_A35x1.bp.trusted_rng**
Random Number Generator unit.
Type: `RandomNumberGenerator`.

**FVP_Base_Cortex_A57x1_A35x1.bp.trusted_watchdog**
ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A57x1_A35x1.bp.trusted_watchdog.clocktimer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x1_A35x1.bp.trusted_watchdog.clocktimer.timer**
A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x1_A35x1.bp.trusted_watchdog.clocktimer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x1_A35x1.bp.trusted_watchdog.clocktimer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x1_A35x1.bp.tzc_400**
TrustZone Address Space Controller.
Type: `TZC_400`.

**FVP_Base_Cortex_A57x1_A35x1.bp.ve_sysregs**
Type: `VE_SysRegs`.

**FVP_Base_Cortex_A57x1_A35x1.bp.vedcc**
Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

**FVP_Base_Cortex_A57x1_A35x1.bp.virtio_net**
VirtioNet device over MMIO transport.
Type: `VirtioNetMMIO`.

**FVP_Base_Cortex_A57x1_A35x1.bp.virtio_net_labeller**
Type: `Labeller`.

**FVP_Base_Cortex_A57x1_A35x1.bp.virtio_rng**
VirtioEntropy device over MMIO transport.
Type: `VirtioEntropyMMIO`.

**FVP_Base_Cortex_A57x1_A35x1.bp.virtioblockdevice**
VirtioBlock device over MMIO transport.
Type: `VirtioBlockMMIO`.

**FVP_Base_Cortex_A57x1_A35x1.bp.virtioblockdevice_labeller**
Type: `Labeller`.

**FVP_Base_Cortex_A57x1_A35x1.bp.virtiop9device**
virtio P9 server.
Type: `VirtioP9Device`.

**FVP_Base_Cortex_A57x1_A35x1.bp.virtiop9device_labeller**
Type: `Labeller`.

**FVP_Base_Cortex_A57x1_A35x1.bp.vis**
Display window for VE using Visualisation library.
Type: `VEVisualisation`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x1_A35x1.bp.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_Base_Cortex_A57x1_A35x1.bp.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x1_A35x1.bp.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x1_A35x1.bp.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x1_A35x1.bp.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x1_A35x1.bp.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x1_A35x1.bp.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x1_A35x1.bp.vram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x1_A35x1.cci400**

Cache Coherent Interconnect for AXI4 ACE.
Type: `CCI400`.

**FVP_Base_Cortex_A57x1_A35x1.clockdivider0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x1_A35x1.clockdivider1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x1_A35x1.cluster0**

ARM Cortex-A57 Cluster CT model.
Type: `Cluster_ARM_Cortex-A57`.

**FVP_Base_Cortex_A57x1_A35x1.cluster0.cpu0**

ARM Cortex-A57 CT model.
Type: `ARM_Cortex-A57`.

**FVP_Base_Cortex_A57x1_A35x1.cluster0.cpu0.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A57x1_A35x1.cluster0.cpu0.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A57x1_A35x1.cluster0.cpu0.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A57x1_A35x1.cluster0.l2_cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A57x1_A35x1.cluster0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A57x1_A35x1.cluster1**

ARM Cortex-A35 Cluster CT model.
Type: `Cluster_ARM_Cortex-A35`.

**FVP_Base_Cortex_A57x1_A35x1.cluster1.cpu0**

ARM Cortex-A35 CT model.
Type: `ARM_Cortex-A35`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x1_A35x1.cluster1.cpu0.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A57x1_A35x1.cluster1.cpu0.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A57x1_A35x1.cluster1.cpu0.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A57x1_A35x1.cluster1.l2_cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A57x1_A35x1.cluster1_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A57x1_A35x1.dapmemlogger**

Bus Logger.
Type: `PVBusLogger`.

**FVP_Base_Cortex_A57x1_A35x1.elfloader**

ELF loader component.
Type: `ElfLoader`.

**FVP_Base_Cortex_A57x1_A35x1.gic_distributor**

GIC Interrupt Redistribution Infrastucture component.
Type: `GIC_IRI`.

**FVP_Base_Cortex_A57x1_A35x1.pctl**

Base Platforms Power Controller.
Type: `Base_PowerController`.

**FVP_Base_Cortex_A57x1_A35x1.pctl.timer_reset**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x1_A35x1.pctl.timer_reset.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x1_A35x1.pctl.timer_reset.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x1_A35x1.pctl.timer_reset.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

## 13.29 FVP_Base_Cortex-A57x1-A53x1

List of instances in FVP_Base_Cortex-A57x1-A53x1.

### FVP_Base_Cortex-A57x1-A53x1 instances

**FVP_Base_Cortex_A57x1_A53x1**

Base Platform Compute Subsystem for ARMCortexA57x1CT and ARMCortexA53x1CT.
Type: `FVP_Base_Cortex_A57x1_A53x1`.

**FVP_Base_Cortex_A57x1_A53x1.bp**

Peripherals and address map for the Base Platform.
Type: `BasePlatformPeripherals`.

**FVP_Base_Cortex_A57x1_A53x1.bp.Timer_0_1**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Cortex_A57x1_A53x1.bp.Timer_0_1.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x1_A53x1.bp.Timer_0_1.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x1_A53x1.bp.Timer_0_1.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A57x1_A53x1.bp.Timer_0_1.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x1_A53x1.bp.Timer_2_3**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Cortex_A57x1_A53x1.bp.Timer_2_3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x1_A53x1.bp.Timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x1_A53x1.bp.Timer_2_3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A57x1_A53x1.bp.Timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A57x1_A53x1.bp.ap_refclk**

ARM Generic Timer.
Type: `MemoryMappedGenericTimer`.

**FVP_Base_Cortex_A57x1_A53x1.bp.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**FVP_Base_Cortex_A57x1_A53x1.bp.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x1_A53x1.bp.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x1_A53x1.bp.clock300MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x1_A53x1.bp.clock32KHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x1_A53x1.bp.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x1_A53x1.bp.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x1_A53x1.bp.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x1_A53x1.bp.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x1_A53x1.bp.dmc**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x1_A53x1.bp.dmc_phy**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x1_A53x1.bp.dummy_local_dap_rom**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x1_A53x1.bp.dummy_ram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x1_A53x1.bp.dummy_usb**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x1_A53x1.bp.exclusive_monitor**
Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_Base_Cortex_A57x1_A53x1.bp.flash0**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A57x1_A53x1.bp.flash1**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A57x1_A53x1.bp.flashloader0**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A57x1_A53x1.bp.flashloader1**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A57x1_A53x1.bp.generic_watchdog**
ARM Generic Watchdog.
Type: `MemoryMappedGenericWatchdog`.

**FVP_Base_Cortex_A57x1_A53x1.bp.hdlcd0**
ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).
Type: `PL370_HDLCD`.

**FVP_Base_Cortex_A57x1_A53x1.bp.hdlcd0.timer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x1_A53x1.bp.hdlcd0.timer.timer**
A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x1_A53x1.bp.hdlcd0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x1_A53x1.bp.hdlcd0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x1_A53x1.bp.hdlcd0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A57x1_A53x1.bp.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_Base_Cortex_A57x1_A53x1.bp.mmc**

Generic Multimedia Card.
Type: `MMC`.

**FVP_Base_Cortex_A57x1_A53x1.bp.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x1_A53x1.bp.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x1_A53x1.bp.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x1_A53x1.bp.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x1_A53x1.bp.mpe**

Encrypt memory transactions for each encryption context with an independent key to prevent mismatch access.
Type: `PVMemoryProtectionEngine`.

**FVP_Base_Cortex_A57x1_A53x1.bp.nontrustedrom**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A57x1_A53x1.bp.nontrustedromloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A57x1_A53x1.bp.ns_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x1_A53x1.bp.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A57x1_A53x1.bp.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x1_A53x1.bp.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x1_A53x1.bp.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x1_A53x1.bp.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x1_A53x1.bp.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x1_A53x1.bp.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A57x1_A53x1.bp.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x1_A53x1.bp.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x1_A53x1.bp.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x1_A53x1.bp.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x1_A53x1.bp.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x1_A53x1.bp.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A57x1_A53x1.bp.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x1_A53x1.bp.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x1_A53x1.bp.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x1_A53x1.bp.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x1_A53x1.bp.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x1_A53x1.bp.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A57x1_A53x1.bp.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x1_A53x1.bp.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x1_A53x1.bp.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x1_A53x1.bp.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x1_A53x1.bp.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x1_A53x1.bp.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

**FVP_Base_Cortex_A57x1_A53x1.bp.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x1_A53x1.bp.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x1_A53x1.bp.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x1_A53x1.bp.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x1_A53x1.bp.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_Base_Cortex_A57x1_A53x1.bp.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x1_A53x1.bp.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x1_A53x1.bp.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x1_A53x1.bp.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x1_A53x1.bp.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A57x1_A53x1.bp.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x1_A53x1.bp.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A57x1_A53x1.bp.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x1_A53x1.bp.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_Base_Cortex_A57x1_A53x1.bp.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

**FVP_Base_Cortex_A57x1_A53x1.bp.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x1_A53x1.bp.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x1_A53x1.bp.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x1_A53x1.bp.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x1_A53x1.bp.pl111_clcd_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A57x1_A53x1.bp.pl180_mci**

ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

**FVP_Base_Cortex_A57x1_A53x1.bp.ps2keyboard**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x1_A53x1.bp.ps2keyboard.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x1_A53x1.bp.ps2keyboard.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x1_A53x1.bp.ps2keyboard.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x1_A53x1.bp.ps2keyboard.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x1_A53x1.bp.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

**FVP_Base_Cortex_A57x1_A53x1.bp.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x1_A53x1.bp.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x1_A53x1.bp.ps2mouse.ps2_clocktimer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x1_A53x1.bp.ps2mouse.ps2_clocktimer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x1_A53x1.bp.psram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x1_A53x1.bp.refcounter**
Memory Mapped Counter Module for Generic Timers.
Type: `MemoryMappedCounterModule`.

**FVP_Base_Cortex_A57x1_A53x1.bp.reset_or**
Or Gate.
Type: `OrGate`.

**FVP_Base_Cortex_A57x1_A53x1.bp.rl_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x1_A53x1.bp.rt_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x1_A53x1.bp.s_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x1_A53x1.bp.secureDRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x1_A53x1.bp.secureSRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x1_A53x1.bp.secureflash**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A57x1_A53x1.bp.secureflashloader**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x1_A53x1.bp.smsc_91c111**

SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_Base_Cortex_A57x1_A53x1.bp.sp805_wdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A57x1_A53x1.bp.sp805_wdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x1_A53x1.bp.sp805_wdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x1_A53x1.bp.sp805_wdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x1_A53x1.bp.sp805_wdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x1_A53x1.bp.sp810_sysctrl**

Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_Base_Cortex_A57x1_A53x1.bp.sp810_sysctrl.clkdiv_clk0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x1_A53x1.bp.sp810_sysctrl.clkdiv_clk1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x1_A53x1.bp.sp810_sysctrl.clkdiv_clk2**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x1_A53x1.bp.sp810_sysctrl.clkdiv_clk3**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x1_A53x1.bp.sram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x1_A53x1.bp.terminal_0**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A57x1_A53x1.bp.terminal_1**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A57x1_A53x1.bp.terminal_2**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A57x1_A53x1.bp.terminal_3**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A57x1_A53x1.bp.trusted_key_storage**

Trusted Root-Key Storage unit.
Type: `RootKeyStorage`.

**FVP_Base_Cortex_A57x1_A53x1.bp.trusted_nv_counter**

Trusted Non-Volatile Counter unit.
Type: `NonVolatileCounter`.

**FVP_Base_Cortex_A57x1_A53x1.bp.trusted_rng**

Random Number Generator unit.
Type: `RandomNumberGenerator`.

**FVP_Base_Cortex_A57x1_A53x1.bp.trusted_watchdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x1_A53x1.bp.trusted_watchdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x1_A53x1.bp.trusted_watchdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x1_A53x1.bp.trusted_watchdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x1_A53x1.bp.trusted_watchdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x1_A53x1.bp.tzc_400**

TrustZone Address Space Controller.
Type: `TZC_400`.

**FVP_Base_Cortex_A57x1_A53x1.bp.ve_sysregs**

Type: `VE_SysRegs`.

**FVP_Base_Cortex_A57x1_A53x1.bp.vedcc**

Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

**FVP_Base_Cortex_A57x1_A53x1.bp.virtio_net**

VirtioNet device over MMIO transport.
Type: `VirtioNetMMIO`.

**FVP_Base_Cortex_A57x1_A53x1.bp.virtio_net_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A57x1_A53x1.bp.virtio_rng**

VirtioEntropy device over MMIO transport.
Type: `VirtioEntropyMMIO`.

**FVP_Base_Cortex_A57x1_A53x1.bp.virtioblockdevice**

VirtioBlock device over MMIO transport.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `VirtioBlockMMIO`.

**FVP_Base_Cortex_A57x1_A53x1.bp.virtioblockdevice_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A57x1_A53x1.bp.virtiop9device**

virtio P9 server.
Type: `VirtioP9Device`.

**FVP_Base_Cortex_A57x1_A53x1.bp.virtiop9device_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A57x1_A53x1.bp.vis**

Display window for VE using Visualisation library.
Type: `VEVisualisation`.

**FVP_Base_Cortex_A57x1_A53x1.bp.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_Base_Cortex_A57x1_A53x1.bp.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x1_A53x1.bp.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x1_A53x1.bp.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x1_A53x1.bp.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x1_A53x1.bp.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x1_A53x1.bp.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x1_A53x1.bp.vram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x1_A53x1.cci400**

Cache Coherent Interconnect for AXI4 ACE.
Type: `CCI400`.

**FVP_Base_Cortex_A57x1_A53x1.clockdivider0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x1_A53x1.clockdivider1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x1_A53x1.cluster0**

ARM Cortex-A57 Cluster CT model.
Type: `Cluster_ARM_Cortex-A57`.

**FVP_Base_Cortex_A57x1_A53x1.cluster0.cpu0**

ARM Cortex-A57 CT model.
Type: `ARM_Cortex-A57`.

**FVP_Base_Cortex_A57x1_A53x1.cluster0.cpu0.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A57x1_A53x1.cluster0.cpu0.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A57x1_A53x1.cluster0.cpu0.l1icache**

PV Cache.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `PVCache`.

**FVP_Base_Cortex_A57x1_A53x1.cluster0.l2_cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A57x1_A53x1.cluster0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A57x1_A53x1.cluster1**

ARM Cortex-A53 Cluster CT model.
Type: `Cluster_ARM_Cortex-A53`.

**FVP_Base_Cortex_A57x1_A53x1.cluster1.cpu0**

ARM Cortex-A53 CT model.
Type: `ARM_Cortex-A53`.

**FVP_Base_Cortex_A57x1_A53x1.cluster1.cpu0.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A57x1_A53x1.cluster1.cpu0.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A57x1_A53x1.cluster1.cpu0.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A57x1_A53x1.cluster1.l2_cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A57x1_A53x1.cluster1_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A57x1_A53x1.dapmemlogger**

Bus Logger.
Type: `PVBusLogger`.

**FVP_Base_Cortex_A57x1_A53x1.elfloader**

ELF loader component.
Type: `ElfLoader`.

**FVP_Base_Cortex_A57x1_A53x1.gic_distributor**

GIC Interrupt Redistribution Infrastucture component.
Type: `GIC_IRI`.

**FVP_Base_Cortex_A57x1_A53x1.pctl**

Base Platforms Power Controller.
Type: `Base_PowerController`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x1_A53x1.pctl.timer_reset**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x1_A53x1.pctl.timer_reset.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x1_A53x1.pctl.timer_reset.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x1_A53x1.pctl.timer_reset.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

# 13.30 FVP_Base_Cortex-A57x2

List of instances in FVP_Base_Cortex-A57x2.

## FVP_Base_Cortex-A57x2 instances

**FVP_Base_Cortex_A57x2**

Base Platform Compute Subsystem for ARMCortexA57x2CT.
Type: `FVP_Base_Cortex_A57x2`.

**FVP_Base_Cortex_A57x2.bp**

Peripherals and address map for the Base Platform.
Type: `BasePlatformPeripherals`.

**FVP_Base_Cortex_A57x2.bp.Timer_0_1**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Cortex_A57x2.bp.Timer_0_1.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x2.bp.Timer_0_1.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x2.bp.Timer_0_1.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A57x2.bp.Timer_0_1.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A57x2.bp.Timer_2_3**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Cortex_A57x2.bp.Timer_2_3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x2.bp.Timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x2.bp.Timer_2_3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A57x2.bp.Timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A57x2.bp.ap_refclk**

ARM Generic Timer.
Type: `MemoryMappedGenericTimer`.

**FVP_Base_Cortex_A57x2.bp.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x2.bp.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x2.bp.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x2.bp.clock300MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x2.bp.clock32KHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x2.bp.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x2.bp.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x2.bp.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x2.bp.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x2.bp.dmc**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x2.bp.dmc_phy**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x2.bp.dummy_local_dap_rom**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x2.bp.dummy_ram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x2.bp.dummy_usb**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x2.bp.exclusive_monitor**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_Base_Cortex_A57x2.bp.flash0**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A57x2.bp.flash1**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A57x2.bp.flashloader0**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A57x2.bp.flashloader1**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A57x2.bp.generic_watchdog**

ARM Generic Watchdog.
Type: `MemoryMappedGenericWatchdog`.

**FVP_Base_Cortex_A57x2.bp.hdlcd0**

ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `PL370_HDLCD`.

**FVP_Base_Cortex_A57x2.bp.hdlcd0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x2.bp.hdlcd0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x2.bp.hdlcd0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x2.bp.hdlcd0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x2.bp.hdlcd0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A57x2.bp.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_Base_Cortex_A57x2.bp.mmc**

Generic Multimedia Card.
Type: `MMC`.

**FVP_Base_Cortex_A57x2.bp.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x2.bp.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x2.bp.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x2.bp.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x2.bp.mpe**

Encrypt memory transactions for each encryption context with an independent key to prevent mismatch access.
Type: `PVMemoryProtectionEngine`.

**FVP_Base_Cortex_A57x2.bp.nontrustedrom**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A57x2.bp.nontrustedromloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A57x2.bp.ns_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x2.bp.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A57x2.bp.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x2.bp.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x2.bp.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x2.bp.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x2.bp.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x2.bp.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A57x2.bp.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x2.bp.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x2.bp.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x2.bp.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x2.bp.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x2.bp.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A57x2.bp.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x2.bp.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x2.bp.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x2.bp.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x2.bp.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x2.bp.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x2.bp.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x2.bp.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x2.bp.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x2.bp.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x2.bp.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x2.bp.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

**FVP_Base_Cortex_A57x2.bp.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x2.bp.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x2.bp.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x2.bp.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x2.bp.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_Base_Cortex_A57x2.bp.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x2.bp.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x2.bp.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x2.bp.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x2.bp.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A57x2.bp.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x2.bp.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A57x2.bp.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x2.bp.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_Base_Cortex_A57x2.bp.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

**FVP_Base_Cortex_A57x2.bp.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x2.bp.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x2.bp.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x2.bp.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x2.bp.pl111_clcd_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A57x2.bp.pl180_mci**

ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

**FVP_Base_Cortex_A57x2.bp.ps2keyboard**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

**FVP_Base_Cortex_A57x2.bp.ps2keyboard.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x2.bp.ps2keyboard.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x2.bp.ps2keyboard.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x2.bp.ps2keyboard.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x2.bp.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `PS2Mouse`.

**FVP_Base_Cortex_A57x2.bp.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x2.bp.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x2.bp.ps2mouse.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x2.bp.ps2mouse.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x2.bp.psram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x2.bp.refcounter**

Memory Mapped Counter Module for Generic Timers.
Type: `MemoryMappedCounterModule`.

**FVP_Base_Cortex_A57x2.bp.reset_or**

Or Gate.
Type: `OrGate`.

**FVP_Base_Cortex_A57x2.bp.rl_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x2.bp.rt_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x2.bp.s_dram**

RAM device, can be dynamic or static ram.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `RAMDevice`.

**FVP_Base_Cortex_A57x2.bp.secureDRAM**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x2.bp.secureSRAM**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x2.bp.secureflash**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A57x2.bp.secureflashloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A57x2.bp.smsc_91c111**

SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_Base_Cortex_A57x2.bp.sp805_wdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A57x2.bp.sp805_wdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x2.bp.sp805_wdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x2.bp.sp805_wdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x2.bp.sp805_wdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x2.bp.sp810_sysctrl**

Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_Base_Cortex_A57x2.bp.sp810_sysctrl.clkdiv_clk0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x2.bp.sp810_sysctrl.clkdiv_clk1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x2.bp.sp810_sysctrl.clkdiv_clk2**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x2.bp.sp810_sysctrl.clkdiv_clk3**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x2.bp.sram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x2.bp.terminal_0**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A57x2.bp.terminal_1**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A57x2.bp.terminal_2**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A57x2.bp.terminal_3**

Telnet terminal interface.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `TelnetTerminal`.

**FVP_Base_Cortex_A57x2.bp.trusted_key_storage**

Trusted Root-Key Storage unit.
Type: `RootKeyStorage`.

**FVP_Base_Cortex_A57x2.bp.trusted_nv_counter**

Trusted Non-Volatile Counter unit.
Type: `NonVolatileCounter`.

**FVP_Base_Cortex_A57x2.bp.trusted_rng**

Random Number Generator unit.
Type: `RandomNumberGenerator`.

**FVP_Base_Cortex_A57x2.bp.trusted_watchdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A57x2.bp.trusted_watchdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x2.bp.trusted_watchdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x2.bp.trusted_watchdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x2.bp.trusted_watchdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x2.bp.tzc_400**

TrustZone Address Space Controller.
Type: `TZC_400`.

**FVP_Base_Cortex_A57x2.bp.ve_sysregs**

Type: `VE_SysRegs`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x2.bp.vedcc**

Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

**FVP_Base_Cortex_A57x2.bp.virtio_net**

VirtioNet device over MMIO transport.
Type: `VirtioNetMMIO`.

**FVP_Base_Cortex_A57x2.bp.virtio_net_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A57x2.bp.virtio_rng**

VirtioEntropy device over MMIO transport.
Type: `VirtioEntropyMMIO`.

**FVP_Base_Cortex_A57x2.bp.virtioblockdevice**

VirtioBlock device over MMIO transport.
Type: `VirtioBlockMMIO`.

**FVP_Base_Cortex_A57x2.bp.virtioblockdevice_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A57x2.bp.virtiop9device**

virtio P9 server.
Type: `VirtioP9Device`.

**FVP_Base_Cortex_A57x2.bp.virtiop9device_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A57x2.bp.vis**

Display window for VE using Visualisation library.
Type: `VEVisualisation`.

**FVP_Base_Cortex_A57x2.bp.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_Base_Cortex_A57x2.bp.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x2.bp.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x2.bp.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x2.bp.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x2.bp.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x2.bp.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x2.bp.vram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x2.cci400**

Cache Coherent Interconnect for AXI4 ACE.
Type: `CCI400`.

**FVP_Base_Cortex_A57x2.clockdivider0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x2.cluster0**

ARM Cortex-A57 Cluster CT model.
Type: `Cluster_ARM_Cortex-A57`.

**FVP_Base_Cortex_A57x2.cluster0.cpu0**

ARM Cortex-A57 CT model.
Type: `ARM_Cortex-A57`.

**FVP_Base_Cortex_A57x2.cluster0.cpu0.dtlb**

TLB - instruction, data or unified.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `TlbCadi`.

**FVP_Base_Cortex_A57x2.cluster0.cpu0.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A57x2.cluster0.cpu0.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A57x2.cluster0.cpu1**

ARM Cortex-A57 CT model.
Type: `ARM_Cortex-A57`.

**FVP_Base_Cortex_A57x2.cluster0.cpu1.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A57x2.cluster0.cpu1.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A57x2.cluster0.cpu1.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A57x2.cluster0.l2_cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A57x2.cluster0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A57x2.dapmemlogger**

Bus Logger.
Type: `PVBusLogger`.

**FVP_Base_Cortex_A57x2.elfloader**

ELF loader component.
Type: `ElfLoader`.

**FVP_Base_Cortex_A57x2.gic_distributor**

GIC Interrupt Redistribution Infrastucture component.
Type: `GIC_IRI`.

**FVP_Base_Cortex_A57x2.pctl**

Base Platforms Power Controller.
Type: `Base_PowerController`.

**FVP_Base_Cortex_A57x2.pctl.timer_reset**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**`FVP_Base_Cortex_A57x2.pctl.timer_reset.timer`**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**`FVP_Base_Cortex_A57x2.pctl.timer_reset.timer.thread`**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**`FVP_Base_Cortex_A57x2.pctl.timer_reset.timer.thread_event`**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

## 13.31 FVP_Base_Cortex-A57x2-A35x4

List of instances in FVP_Base_Cortex-A57x2-A35x4.

### FVP_Base_Cortex-A57x2-A35x4 instances

**`FVP_Base_Cortex_A57x2_A35x4`**

Base Platform Compute Subsystem for ARMCortexA57x2CT and ARMCortexA35x4CT.
Type: `FVP_Base_Cortex_A57x2_A35x4`.

**`FVP_Base_Cortex_A57x2_A35x4.bp`**

Peripherals and address map for the Base Platform.
Type: `BasePlatformPeripherals`.

**`FVP_Base_Cortex_A57x2_A35x4.bp.Timer_0_1`**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**`FVP_Base_Cortex_A57x2_A35x4.bp.Timer_0_1.clk_div0`**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x2_A35x4.bp.Timer_0_1.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x2_A35x4.bp.Timer_0_1.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A57x2_A35x4.bp.Timer_0_1.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A57x2_A35x4.bp.Timer_2_3**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Cortex_A57x2_A35x4.bp.Timer_2_3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x2_A35x4.bp.Timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x2_A35x4.bp.Timer_2_3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A57x2_A35x4.bp.Timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A57x2_A35x4.bp.ap_refclk**

ARM Generic Timer.
Type: `MemoryMappedGenericTimer`.

**FVP_Base_Cortex_A57x2_A35x4.bp.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**FVP_Base_Cortex_A57x2_A35x4.bp.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x2_A35x4.bp.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x2_A35x4.bp.clock300MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x2_A35x4.bp.clock32KHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x2_A35x4.bp.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x2_A35x4.bp.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x2_A35x4.bp.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x2_A35x4.bp.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ClockDivider`.

**FVP_Base_Cortex_A57x2_A35x4.bp.dmc**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x2_A35x4.bp.dmc_phy**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x2_A35x4.bp.dummy_local_dap_rom**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x2_A35x4.bp.dummy_ram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x2_A35x4.bp.dummy_usb**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x2_A35x4.bp.exclusive_monitor**
Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_Base_Cortex_A57x2_A35x4.bp.flash0**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A57x2_A35x4.bp.flash1**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A57x2_A35x4.bp.flashloader0**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A57x2_A35x4.bp.flashloader1**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A57x2_A35x4.bp.generic_watchdog**
ARM Generic Watchdog.
Type: `MemoryMappedGenericWatchdog`.

**FVP_Base_Cortex_A57x2_A35x4.bp.hdlcd0**
ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).
Type: `PL370_HDLCD`.

**FVP_Base_Cortex_A57x2_A35x4.bp.hdlcd0.timer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x2_A35x4.bp.hdlcd0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x2_A35x4.bp.hdlcd0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x2_A35x4.bp.hdlcd0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x2_A35x4.bp.hdlcd0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A57x2_A35x4.bp.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_Base_Cortex_A57x2_A35x4.bp.mmc**

Generic Multimedia Card.
Type: `MMC`.

**FVP_Base_Cortex_A57x2_A35x4.bp.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x2_A35x4.bp.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x2_A35x4.bp.mmc.timer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x2_A35x4.bp.mmc.timer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x2_A35x4.bp.mpe**
Encrypt memory transactions for each encryption context with an independent key to prevent mismatch access.
Type: `PVMemoryProtectionEngine`.

**FVP_Base_Cortex_A57x2_A35x4.bp.nontrustedrom**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A57x2_A35x4.bp.nontrustedromloader**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A57x2_A35x4.bp.ns_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x2_A35x4.bp.pl011_uart0**
ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A57x2_A35x4.bp.pl011_uart0.clk_divider**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x2_A35x4.bp.pl011_uart0.timer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x2_A35x4.bp.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x2_A35x4.bp.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x2_A35x4.bp.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x2_A35x4.bp.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A57x2_A35x4.bp.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x2_A35x4.bp.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x2_A35x4.bp.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x2_A35x4.bp.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x2_A35x4.bp.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x2_A35x4.bp.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A57x2_A35x4.bp.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x2_A35x4.bp.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x2_A35x4.bp.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x2_A35x4.bp.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x2_A35x4.bp.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x2_A35x4.bp.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A57x2_A35x4.bp.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x2_A35x4.bp.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x2_A35x4.bp.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x2_A35x4.bp.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x2_A35x4.bp.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x2_A35x4.bp.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

**FVP_Base_Cortex_A57x2_A35x4.bp.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x2_A35x4.bp.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x2_A35x4.bp.pl031_rtc.timer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x2_A35x4.bp.pl031_rtc.timer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x2_A35x4.bp.pl041_aaci**
ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_Base_Cortex_A57x2_A35x4.bp.pl041_aaci.timer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x2_A35x4.bp.pl041_aaci.timer.timer**
A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x2_A35x4.bp.pl041_aaci.timer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x2_A35x4.bp.pl041_aaci.timer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x2_A35x4.bp.pl050_kmi0**
ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A57x2_A35x4.bp.pl050_kmi0.clk_divider**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x2_A35x4.bp.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A57x2_A35x4.bp.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x2_A35x4.bp.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_Base_Cortex_A57x2_A35x4.bp.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

**FVP_Base_Cortex_A57x2_A35x4.bp.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x2_A35x4.bp.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x2_A35x4.bp.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x2_A35x4.bp.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x2_A35x4.bp.pl111_clcd_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A57x2_A35x4.bp.pl180_mci**

ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

**FVP_Base_Cortex_A57x2_A35x4.bp.ps2keyboard**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

**FVP_Base_Cortex_A57x2_A35x4.bp.ps2keyboard.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x2_A35x4.bp.ps2keyboard.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x2_A35x4.bp.ps2keyboard.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x2_A35x4.bp.ps2keyboard.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x2_A35x4.bp.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

**FVP_Base_Cortex_A57x2_A35x4.bp.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x2_A35x4.bp.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x2_A35x4.bp.ps2mouse.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x2_A35x4.bp.ps2mouse.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x2_A35x4.bp.psram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x2_A35x4.bp.refcounter**

Memory Mapped Counter Module for Generic Timers.
Type: `MemoryMappedCounterModule`.

**FVP_Base_Cortex_A57x2_A35x4.bp.reset_or**

Or Gate.
Type: `OrGate`.

**FVP_Base_Cortex_A57x2_A35x4.bp.rl_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x2_A35x4.bp.rt_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x2_A35x4.bp.s_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x2_A35x4.bp.secureDRAM**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x2_A35x4.bp.secureSRAM**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x2_A35x4.bp.secureflash**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A57x2_A35x4.bp.secureflashloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A57x2_A35x4.bp.smsc_91c111**

SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_Base_Cortex_A57x2_A35x4.bp.sp805_wdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A57x2_A35x4.bp.sp805_wdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x2_A35x4.bp.sp805_wdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x2_A35x4.bp.sp805_wdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x2_A35x4.bp.sp805_wdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x2_A35x4.bp.sp810_sysctrl**

Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x2_A35x4.bp.sp810_sysctrl.clkdiv_clk0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x2_A35x4.bp.sp810_sysctrl.clkdiv_clk1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x2_A35x4.bp.sp810_sysctrl.clkdiv_clk2**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x2_A35x4.bp.sp810_sysctrl.clkdiv_clk3**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x2_A35x4.bp.sram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x2_A35x4.bp.terminal_0**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A57x2_A35x4.bp.terminal_1**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A57x2_A35x4.bp.terminal_2**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A57x2_A35x4.bp.terminal_3**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A57x2_A35x4.bp.trusted_key_storage**

Trusted Root-Key Storage unit.
Type: `RootKeyStorage`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x2_A35x4.bp.trusted_nv_counter**

Trusted Non-Volatile Counter unit.
Type: `NonVolatileCounter`.

**FVP_Base_Cortex_A57x2_A35x4.bp.trusted_rng**

Random Number Generator unit.
Type: `RandomNumberGenerator`.

**FVP_Base_Cortex_A57x2_A35x4.bp.trusted_watchdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A57x2_A35x4.bp.trusted_watchdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x2_A35x4.bp.trusted_watchdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x2_A35x4.bp.trusted_watchdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x2_A35x4.bp.trusted_watchdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x2_A35x4.bp.tzc_400**

TrustZone Address Space Controller.
Type: `TZC_400`.

**FVP_Base_Cortex_A57x2_A35x4.bp.ve_sysregs**

Type: `VE_SysRegs`.

**FVP_Base_Cortex_A57x2_A35x4.bp.vedcc**

Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x2_A35x4.bp.virtio_net**

VirtioNet device over MMIO transport.
Type: `VirtioNetMMIO`.

**FVP_Base_Cortex_A57x2_A35x4.bp.virtio_net_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A57x2_A35x4.bp.virtio_rng**

VirtioEntropy device over MMIO transport.
Type: `VirtioEntropyMMIO`.

**FVP_Base_Cortex_A57x2_A35x4.bp.virtioblockdevice**

VirtioBlock device over MMIO transport.
Type: `VirtioBlockMMIO`.

**FVP_Base_Cortex_A57x2_A35x4.bp.virtioblockdevice_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A57x2_A35x4.bp.virtiop9device**

virtio P9 server.
Type: `VirtioP9Device`.

**FVP_Base_Cortex_A57x2_A35x4.bp.virtiop9device_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A57x2_A35x4.bp.vis**

Display window for VE using Visualisation library.
Type: `VEVisualisation`.

**FVP_Base_Cortex_A57x2_A35x4.bp.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_Base_Cortex_A57x2_A35x4.bp.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x2_A35x4.bp.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x2_A35x4.bp.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x2_A35x4.bp.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x2_A35x4.bp.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x2_A35x4.bp.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x2_A35x4.bp.vram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x2_A35x4.cci400**

Cache Coherent Interconnect for AXI4 ACE.
Type: `CCI400`.

**FVP_Base_Cortex_A57x2_A35x4.clockdivider0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x2_A35x4.clockdivider1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x2_A35x4.cluster0**

ARM Cortex-A57 Cluster CT model.
Type: `Cluster_ARM_Cortex-A57`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x2_A35x4.cluster0.cpu0**
ARM Cortex-A57 CT model.
Type: `ARM_Cortex-A57`.

**FVP_Base_Cortex_A57x2_A35x4.cluster0.cpu0.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A57x2_A35x4.cluster0.cpu0.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A57x2_A35x4.cluster0.cpu0.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A57x2_A35x4.cluster0.cpu1**
ARM Cortex-A57 CT model.
Type: `ARM_Cortex-A57`.

**FVP_Base_Cortex_A57x2_A35x4.cluster0.cpu1.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A57x2_A35x4.cluster0.cpu1.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A57x2_A35x4.cluster0.cpu1.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A57x2_A35x4.cluster0.l2_cache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A57x2_A35x4.cluster0_labeller**
Type: `Labeller`.

**FVP_Base_Cortex_A57x2_A35x4.cluster1**
ARM Cortex-A35 Cluster CT model.
Type: `Cluster_ARM_Cortex-A35`.

**FVP_Base_Cortex_A57x2_A35x4.cluster1.cpu0**
ARM Cortex-A35 CT model.
Type: `ARM_Cortex-A35`.

**FVP_Base_Cortex_A57x2_A35x4.cluster1.cpu0.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A57x2_A35x4.cluster1.cpu0.l1dcache**
PV Cache.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `PVCache`.

**FVP_Base_Cortex_A57x2_A35x4.cluster1.cpu0.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A57x2_A35x4.cluster1.cpu1**

ARM Cortex-A35 CT model.
Type: `ARM_Cortex-A35`.

**FVP_Base_Cortex_A57x2_A35x4.cluster1.cpu1.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A57x2_A35x4.cluster1.cpu1.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A57x2_A35x4.cluster1.cpu1.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A57x2_A35x4.cluster1.cpu2**

ARM Cortex-A35 CT model.
Type: `ARM_Cortex-A35`.

**FVP_Base_Cortex_A57x2_A35x4.cluster1.cpu2.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A57x2_A35x4.cluster1.cpu2.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A57x2_A35x4.cluster1.cpu2.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A57x2_A35x4.cluster1.cpu3**

ARM Cortex-A35 CT model.
Type: `ARM_Cortex-A35`.

**FVP_Base_Cortex_A57x2_A35x4.cluster1.cpu3.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A57x2_A35x4.cluster1.cpu3.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A57x2_A35x4.cluster1.cpu3.l1icache**

PV Cache.
Type: `PVCache`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x2_A35x4.cluster1.l2_cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A57x2_A35x4.cluster1_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A57x2_A35x4.dapmemlogger**

Bus Logger.
Type: `PVBusLogger`.

**FVP_Base_Cortex_A57x2_A35x4.elfloader**

ELF loader component.
Type: `ElfLoader`.

**FVP_Base_Cortex_A57x2_A35x4.gic_distributor**

GIC Interrupt Redistribution Infrastucture component.
Type: `GIC_IRI`.

**FVP_Base_Cortex_A57x2_A35x4.pctl**

Base Platforms Power Controller.
Type: `Base_PowerController`.

**FVP_Base_Cortex_A57x2_A35x4.pctl.timer_reset**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x2_A35x4.pctl.timer_reset.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x2_A35x4.pctl.timer_reset.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x2_A35x4.pctl.timer_reset.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

# 13.32 FVP_Base_Cortex-A57x2-A53x4

List of instances in FVP_Base_Cortex-A57x2-A53x4.

## FVP_Base_Cortex-A57x2-A53x4 instances

**FVP_Base_Cortex_A57x2_A53x4**

Base Platform Compute Subsystem for ARMCortexA57x2CT and ARMCortexA53x4CT.
Type: `FVP_Base_Cortex_A57x2_A53x4`.

**FVP_Base_Cortex_A57x2_A53x4.bp**

Peripherals and address map for the Base Platform.
Type: `BasePlatformPeripherals`.

**FVP_Base_Cortex_A57x2_A53x4.bp.Timer_0_1**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Cortex_A57x2_A53x4.bp.Timer_0_1.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x2_A53x4.bp.Timer_0_1.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x2_A53x4.bp.Timer_0_1.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A57x2_A53x4.bp.Timer_0_1.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A57x2_A53x4.bp.Timer_2_3**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Cortex_A57x2_A53x4.bp.Timer_2_3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x2_A53x4.bp.Timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x2_A53x4.bp.Timer_2_3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A57x2_A53x4.bp.Timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A57x2_A53x4.bp.ap_refclk**

ARM Generic Timer.
Type: `MemoryMappedGenericTimer`.

**FVP_Base_Cortex_A57x2_A53x4.bp.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**FVP_Base_Cortex_A57x2_A53x4.bp.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x2_A53x4.bp.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x2_A53x4.bp.clock300MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x2_A53x4.bp.clock32KHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x2_A53x4.bp.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x2_A53x4.bp.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x2_A53x4.bp.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x2_A53x4.bp.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x2_A53x4.bp.dmc**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x2_A53x4.bp.dmc_phy**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x2_A53x4.bp.dummy_local_dap_rom**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x2_A53x4.bp.dummy_ram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x2_A53x4.bp.dummy_usb**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x2_A53x4.bp.exclusive_monitor**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x2_A53x4.bp.flash0**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A57x2_A53x4.bp.flash1**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A57x2_A53x4.bp.flashloader0**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A57x2_A53x4.bp.flashloader1**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A57x2_A53x4.bp.generic_watchdog**

ARM Generic Watchdog.
Type: `MemoryMappedGenericWatchdog`.

**FVP_Base_Cortex_A57x2_A53x4.bp.hdlcd0**

ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).
Type: `PL370_HDLCD`.

**FVP_Base_Cortex_A57x2_A53x4.bp.hdlcd0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x2_A53x4.bp.hdlcd0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x2_A53x4.bp.hdlcd0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x2_A53x4.bp.hdlcd0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x2_A53x4.bp.hdlcd0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A57x2_A53x4.bp.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_Base_Cortex_A57x2_A53x4.bp.mmc**

Generic Multimedia Card.
Type: `MMC`.

**FVP_Base_Cortex_A57x2_A53x4.bp.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x2_A53x4.bp.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x2_A53x4.bp.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x2_A53x4.bp.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x2_A53x4.bp.mpe**

Encrypt memory transactions for each encryption context with an independent key to prevent mismatch access.
Type: `PVMemoryProtectionEngine`.

**FVP_Base_Cortex_A57x2_A53x4.bp.nontrustedrom**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A57x2_A53x4.bp.nontrustedromloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x2_A53x4.bp.ns_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x2_A53x4.bp.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A57x2_A53x4.bp.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x2_A53x4.bp.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x2_A53x4.bp.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x2_A53x4.bp.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x2_A53x4.bp.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x2_A53x4.bp.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A57x2_A53x4.bp.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x2_A53x4.bp.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x2_A53x4.bp.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x2_A53x4.bp.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x2_A53x4.bp.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x2_A53x4.bp.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A57x2_A53x4.bp.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x2_A53x4.bp.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x2_A53x4.bp.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x2_A53x4.bp.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x2_A53x4.bp.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x2_A53x4.bp.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A57x2_A53x4.bp.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x2_A53x4.bp.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x2_A53x4.bp.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x2_A53x4.bp.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x2_A53x4.bp.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x2_A53x4.bp.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

**FVP_Base_Cortex_A57x2_A53x4.bp.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x2_A53x4.bp.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x2_A53x4.bp.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x2_A53x4.bp.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x2_A53x4.bp.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_Base_Cortex_A57x2_A53x4.bp.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x2_A53x4.bp.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x2_A53x4.bp.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x2_A53x4.bp.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x2_A53x4.bp.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A57x2_A53x4.bp.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x2_A53x4.bp.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A57x2_A53x4.bp.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x2_A53x4.bp.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_Base_Cortex_A57x2_A53x4.bp.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x2_A53x4.bp.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x2_A53x4.bp.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x2_A53x4.bp.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x2_A53x4.bp.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x2_A53x4.bp.pl111_clcd_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A57x2_A53x4.bp.pl180_mci**

ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

**FVP_Base_Cortex_A57x2_A53x4.bp.ps2keyboard**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

**FVP_Base_Cortex_A57x2_A53x4.bp.ps2keyboard.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x2_A53x4.bp.ps2keyboard.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x2_A53x4.bp.ps2keyboard.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x2_A53x4.bp.ps2keyboard.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x2_A53x4.bp.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

**FVP_Base_Cortex_A57x2_A53x4.bp.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x2_A53x4.bp.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x2_A53x4.bp.ps2mouse.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x2_A53x4.bp.ps2mouse.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x2_A53x4.bp.psram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x2_A53x4.bp.refcounter**
Memory Mapped Counter Module for Generic Timers.
Type: `MemoryMappedCounterModule`.

**FVP_Base_Cortex_A57x2_A53x4.bp.reset_or**
Or Gate.
Type: `OrGate`.

**FVP_Base_Cortex_A57x2_A53x4.bp.rl_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x2_A53x4.bp.rt_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x2_A53x4.bp.s_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x2_A53x4.bp.secureDRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x2_A53x4.bp.secureSRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x2_A53x4.bp.secureflash**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A57x2_A53x4.bp.secureflashloader**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A57x2_A53x4.bp.smsc_91c111**
SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_Base_Cortex_A57x2_A53x4.bp.sp805_wdog**
ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A57x2_A53x4.bp.sp805_wdog.clocktimer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x2_A53x4.bp.sp805_wdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x2_A53x4.bp.sp805_wdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x2_A53x4.bp.sp805_wdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x2_A53x4.bp.sp810_sysctrl**

Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_Base_Cortex_A57x2_A53x4.bp.sp810_sysctrl.clkdiv_clk0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x2_A53x4.bp.sp810_sysctrl.clkdiv_clk1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x2_A53x4.bp.sp810_sysctrl.clkdiv_clk2**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x2_A53x4.bp.sp810_sysctrl.clkdiv_clk3**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x2_A53x4.bp.sram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x2_A53x4.bp.terminal_0**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A57x2_A53x4.bp.terminal_1**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A57x2_A53x4.bp.terminal_2**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A57x2_A53x4.bp.terminal_3**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A57x2_A53x4.bp.trusted_key_storage**
Trusted Root-Key Storage unit.
Type: `RootKeyStorage`.

**FVP_Base_Cortex_A57x2_A53x4.bp.trusted_nv_counter**
Trusted Non-Volatile Counter unit.
Type: `NonVolatileCounter`.

**FVP_Base_Cortex_A57x2_A53x4.bp.trusted_rng**
Random Number Generator unit.
Type: `RandomNumberGenerator`.

**FVP_Base_Cortex_A57x2_A53x4.bp.trusted_watchdog**
ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A57x2_A53x4.bp.trusted_watchdog.clocktimer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x2_A53x4.bp.trusted_watchdog.clocktimer.timer**
A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x2_A53x4.bp.trusted_watchdog.clocktimer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x2_A53x4.bp.trusted_watchdog.clocktimer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x2_A53x4.bp.tzc_400**
TrustZone Address Space Controller.
Type: `TZC_400`.

**FVP_Base_Cortex_A57x2_A53x4.bp.ve_sysregs**
Type: `VE_SysRegs`.

**FVP_Base_Cortex_A57x2_A53x4.bp.vedcc**
Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

**FVP_Base_Cortex_A57x2_A53x4.bp.virtio_net**
VirtioNet device over MMIO transport.
Type: `VirtioNetMMIO`.

**FVP_Base_Cortex_A57x2_A53x4.bp.virtio_net_labeller**
Type: `Labeller`.

**FVP_Base_Cortex_A57x2_A53x4.bp.virtio_rng**
VirtioEntropy device over MMIO transport.
Type: `VirtioEntropyMMIO`.

**FVP_Base_Cortex_A57x2_A53x4.bp.virtioblockdevice**
VirtioBlock device over MMIO transport.
Type: `VirtioBlockMMIO`.

**FVP_Base_Cortex_A57x2_A53x4.bp.virtioblockdevice_labeller**
Type: `Labeller`.

**FVP_Base_Cortex_A57x2_A53x4.bp.virtiop9device**
virtio P9 server.
Type: `VirtioP9Device`.

**FVP_Base_Cortex_A57x2_A53x4.bp.virtiop9device_labeller**
Type: `Labeller`.

**FVP_Base_Cortex_A57x2_A53x4.bp.vis**
Display window for VE using Visualisation library.
Type: `VEVisualisation`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x2_A53x4.bp.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_Base_Cortex_A57x2_A53x4.bp.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x2_A53x4.bp.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x2_A53x4.bp.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x2_A53x4.bp.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x2_A53x4.bp.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x2_A53x4.bp.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x2_A53x4.bp.vram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x2_A53x4.cci400**

Cache Coherent Interconnect for AXI4 ACE.
Type: `CCI400`.

**FVP_Base_Cortex_A57x2_A53x4.clockdivider0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x2_A53x4.clockdivider1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x2_A53x4.cluster0**

ARM Cortex-A57 Cluster CT model.
Type: `Cluster_ARM_Cortex-A57`.

**FVP_Base_Cortex_A57x2_A53x4.cluster0.cpu0**

ARM Cortex-A57 CT model.
Type: `ARM_Cortex-A57`.

**FVP_Base_Cortex_A57x2_A53x4.cluster0.cpu0.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A57x2_A53x4.cluster0.cpu0.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A57x2_A53x4.cluster0.cpu0.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A57x2_A53x4.cluster0.cpu1**

ARM Cortex-A57 CT model.
Type: `ARM_Cortex-A57`.

**FVP_Base_Cortex_A57x2_A53x4.cluster0.cpu1.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A57x2_A53x4.cluster0.cpu1.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A57x2_A53x4.cluster0.cpu1.l1icache**

PV Cache.
Type: `PVCache`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x2_A53x4.cluster0.l2_cache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A57x2_A53x4.cluster0_labeller**
Type: `Labeller`.

**FVP_Base_Cortex_A57x2_A53x4.cluster1**
ARM Cortex-A53 Cluster CT model.
Type: `Cluster_ARM_Cortex-A53`.

**FVP_Base_Cortex_A57x2_A53x4.cluster1.cpu0**
ARM Cortex-A53 CT model.
Type: `ARM_Cortex-A53`.

**FVP_Base_Cortex_A57x2_A53x4.cluster1.cpu0.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A57x2_A53x4.cluster1.cpu0.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A57x2_A53x4.cluster1.cpu0.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A57x2_A53x4.cluster1.cpu1**
ARM Cortex-A53 CT model.
Type: `ARM_Cortex-A53`.

**FVP_Base_Cortex_A57x2_A53x4.cluster1.cpu1.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A57x2_A53x4.cluster1.cpu1.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A57x2_A53x4.cluster1.cpu1.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A57x2_A53x4.cluster1.cpu2**
ARM Cortex-A53 CT model.
Type: `ARM_Cortex-A53`.

**FVP_Base_Cortex_A57x2_A53x4.cluster1.cpu2.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A57x2_A53x4.cluster1.cpu2.l1dcache**
PV Cache.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `PVCache`.

**FVP_Base_Cortex_A57x2_A53x4.cluster1.cpu2.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A57x2_A53x4.cluster1.cpu3**

ARM Cortex-A53 CT model.
Type: `ARM_Cortex-A53`.

**FVP_Base_Cortex_A57x2_A53x4.cluster1.cpu3.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A57x2_A53x4.cluster1.cpu3.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A57x2_A53x4.cluster1.cpu3.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A57x2_A53x4.cluster1.l2_cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A57x2_A53x4.cluster1_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A57x2_A53x4.dapmemlogger**

Bus Logger.
Type: `PVBusLogger`.

**FVP_Base_Cortex_A57x2_A53x4.elfloader**

ELF loader component.
Type: `ElfLoader`.

**FVP_Base_Cortex_A57x2_A53x4.gic_distributor**

GIC Interrupt Redistribution Infrastucture component.
Type: `GIC_IRI`.

**FVP_Base_Cortex_A57x2_A53x4.pctl**

Base Platforms Power Controller.
Type: `Base_PowerController`.

**FVP_Base_Cortex_A57x2_A53x4.pctl.timer_reset**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ClockTimerThread`.

**`FVP_Base_Cortex_A57x2_A53x4.pctl.timer_reset.timer`**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**`FVP_Base_Cortex_A57x2_A53x4.pctl.timer_reset.timer.thread`**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**`FVP_Base_Cortex_A57x2_A53x4.pctl.timer_reset.timer.thread_event`**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

# 13.33 FVP_Base_Cortex-A57x4

List of instances in FVP_Base_Cortex-A57x4.

## FVP_Base_Cortex-A57x4 instances

**`FVP_Base_Cortex_A57x4`**

Base Platform Compute Subsystem for ARMCortexA57x4CT.
Type: `FVP_Base_Cortex_A57x4`.

**`FVP_Base_Cortex_A57x4.bp`**

Peripherals and address map for the Base Platform.
Type: `BasePlatformPeripherals`.

**`FVP_Base_Cortex_A57x4.bp.Timer_0_1`**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**`FVP_Base_Cortex_A57x4.bp.Timer_0_1.clk_div0`**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**`FVP_Base_Cortex_A57x4.bp.Timer_0_1.clk_div1`**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ClockDivider`.

**FVP_Base_Cortex_A57x4.bp.Timer_0_1.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A57x4.bp.Timer_0_1.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A57x4.bp.Timer_2_3**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Cortex_A57x4.bp.Timer_2_3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x4.bp.Timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x4.bp.Timer_2_3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A57x4.bp.Timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A57x4.bp.ap_refclk**

ARM Generic Timer.
Type: `MemoryMappedGenericTimer`.

**FVP_Base_Cortex_A57x4.bp.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**FVP_Base_Cortex_A57x4.bp.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x4.bp.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x4.bp.clock300MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x4.bp.clock32KHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x4.bp.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x4.bp.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x4.bp.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x4.bp.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x4.bp.dmc**

RAM device, can be dynamic or static ram.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `RAMDevice`.

**FVP_Base_Cortex_A57x4.bp.dmc_phy**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x4.bp.dummy_local_dap_rom**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x4.bp.dummy_ram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x4.bp.dummy_usb**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x4.bp.exclusive_monitor**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_Base_Cortex_A57x4.bp.flash0**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A57x4.bp.flash1**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A57x4.bp.flashloader0**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A57x4.bp.flashloader1**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A57x4.bp.generic_watchdog**

ARM Generic Watchdog.
Type: `MemoryMappedGenericWatchdog`.

**FVP_Base_Cortex_A57x4.bp.hdlcd0**

ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).
Type: `PL370_HDLCD`.

**FVP_Base_Cortex_A57x4.bp.hdlcd0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x4.bp.hdlcd0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x4.bp.hdlcd0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x4.bp.hdlcd0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x4.bp.hdlcd0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A57x4.bp.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_Base_Cortex_A57x4.bp.mmc**

Generic Multimedia Card.
Type: `MMC`.

**FVP_Base_Cortex_A57x4.bp.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x4.bp.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x4.bp.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x4.bp.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x4.bp.mpe**

Encrypt memory transactions for each encryption context with an independent key to prevent mismatch access.
Type: `PVMemoryProtectionEngine`.

**FVP_Base_Cortex_A57x4.bp.nontrustedrom**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A57x4.bp.nontrustedromloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A57x4.bp.ns_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x4.bp.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A57x4.bp.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x4.bp.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x4.bp.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x4.bp.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x4.bp.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x4.bp.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A57x4.bp.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x4.bp.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x4.bp.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x4.bp.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x4.bp.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x4.bp.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A57x4.bp.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x4.bp.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x4.bp.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x4.bp.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x4.bp.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x4.bp.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A57x4.bp.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x4.bp.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x4.bp.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x4.bp.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x4.bp.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x4.bp.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

**FVP_Base_Cortex_A57x4.bp.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x4.bp.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x4.bp.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x4.bp.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x4.bp.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_Base_Cortex_A57x4.bp.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x4.bp.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x4.bp.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x4.bp.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x4.bp.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A57x4.bp.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x4.bp.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A57x4.bp.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x4.bp.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_Base_Cortex_A57x4.bp.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

**FVP_Base_Cortex_A57x4.bp.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x4.bp.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x4.bp.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x4.bp.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x4.bp.pl111_clcd_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A57x4.bp.pl180_mci**

ARM PrimeCell Multimedia Card Interface (PL180).

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `PL180_MCI`.

**FVP_Base_Cortex_A57x4.bp.ps2keyboard**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

**FVP_Base_Cortex_A57x4.bp.ps2keyboard.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x4.bp.ps2keyboard.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x4.bp.ps2keyboard.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x4.bp.ps2keyboard.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x4.bp.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

**FVP_Base_Cortex_A57x4.bp.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x4.bp.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x4.bp.ps2mouse.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x4.bp.ps2mouse.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x4.bp.psram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x4.bp.refcounter**

Memory Mapped Counter Module for Generic Timers.
Type: `MemoryMappedCounterModule`.

**FVP_Base_Cortex_A57x4.bp.reset_or**

Or Gate.
Type: `OrGate`.

**FVP_Base_Cortex_A57x4.bp.rl_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x4.bp.rt_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x4.bp.s_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x4.bp.secureDRAM**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x4.bp.secureSRAM**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x4.bp.secureflash**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x4.bp.secureflashloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A57x4.bp.smsc_91c111**

SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_Base_Cortex_A57x4.bp.sp805_wdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A57x4.bp.sp805_wdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x4.bp.sp805_wdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x4.bp.sp805_wdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x4.bp.sp805_wdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x4.bp.sp810_sysctrl**

Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_Base_Cortex_A57x4.bp.sp810_sysctrl.clkdiv_clk0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x4.bp.sp810_sysctrl.clkdiv_clk1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x4.bp.sp810_sysctrl.clkdiv_clk2**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x4.bp.sp810_sysctrl.clkdiv_clk3**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x4.bp.sram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x4.bp.terminal_0**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A57x4.bp.terminal_1**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A57x4.bp.terminal_2**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A57x4.bp.terminal_3**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A57x4.bp.trusted_key_storage**

Trusted Root-Key Storage unit.
Type: `RootKeyStorage`.

**FVP_Base_Cortex_A57x4.bp.trusted_nv_counter**

Trusted Non-Volatile Counter unit.
Type: `NonVolatileCounter`.

**FVP_Base_Cortex_A57x4.bp.trusted_rng**

Random Number Generator unit.
Type: `RandomNumberGenerator`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x4.bp.trusted_watchdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A57x4.bp.trusted_watchdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x4.bp.trusted_watchdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x4.bp.trusted_watchdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x4.bp.trusted_watchdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x4.bp.tzc_400**

TrustZone Address Space Controller.
Type: `TZC_400`.

**FVP_Base_Cortex_A57x4.bp.ve_sysregs**

Type: `VE_SysRegs`.

**FVP_Base_Cortex_A57x4.bp.vedcc**

Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

**FVP_Base_Cortex_A57x4.bp.virtio_net**

VirtioNet device over MMIO transport.
Type: `VirtioNetMMIO`.

**FVP_Base_Cortex_A57x4.bp.virtio_net_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A57x4.bp.virtio_rng**

VirtioEntropy device over MMIO transport.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `VirtioEntropyMMIO`.

**FVP_Base_Cortex_A57x4.bp.virtioblockdevice**

VirtioBlock device over MMIO transport.
Type: `VirtioBlockMMIO`.

**FVP_Base_Cortex_A57x4.bp.virtioblockdevice_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A57x4.bp.virtiop9device**

virtio P9 server.
Type: `VirtioP9Device`.

**FVP_Base_Cortex_A57x4.bp.virtiop9device_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A57x4.bp.vis**

Display window for VE using Visualisation library.
Type: `VEVisualisation`.

**FVP_Base_Cortex_A57x4.bp.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_Base_Cortex_A57x4.bp.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x4.bp.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x4.bp.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x4.bp.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x4.bp.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x4.bp.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x4.bp.vram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x4.cci400**

Cache Coherent Interconnect for AXI4 ACE.
Type: `CCI400`.

**FVP_Base_Cortex_A57x4.clockdivider0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x4.cluster0**

ARM Cortex-A57 Cluster CT model.
Type: `Cluster_ARM_Cortex-A57`.

**FVP_Base_Cortex_A57x4.cluster0.cpu0**

ARM Cortex-A57 CT model.
Type: `ARM_Cortex-A57`.

**FVP_Base_Cortex_A57x4.cluster0.cpu0.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A57x4.cluster0.cpu0.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A57x4.cluster0.cpu0.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A57x4.cluster0.cpu1**

ARM Cortex-A57 CT model.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ARM_Cortex-A57`.

**FVP_Base_Cortex_A57x4.cluster0.cpu1.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A57x4.cluster0.cpu1.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A57x4.cluster0.cpu1.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A57x4.cluster0.cpu2**

ARM Cortex-A57 CT model.
Type: `ARM_Cortex-A57`.

**FVP_Base_Cortex_A57x4.cluster0.cpu2.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A57x4.cluster0.cpu2.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A57x4.cluster0.cpu2.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A57x4.cluster0.cpu3**

ARM Cortex-A57 CT model.
Type: `ARM_Cortex-A57`.

**FVP_Base_Cortex_A57x4.cluster0.cpu3.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A57x4.cluster0.cpu3.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A57x4.cluster0.cpu3.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A57x4.cluster0.l2_cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A57x4.cluster0_labeller**

Type: `Labeller`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x4.dapmemlogger**

Bus Logger.
Type: `PVBusLogger`.

**FVP_Base_Cortex_A57x4.elfloader**

ELF loader component.
Type: `ElfLoader`.

**FVP_Base_Cortex_A57x4.gic_distributor**

GIC Interrupt Redistribution Infrastucture component.
Type: `GIC_IRI`.

**FVP_Base_Cortex_A57x4.pctl**

Base Platforms Power Controller.
Type: `Base_PowerController`.

**FVP_Base_Cortex_A57x4.pctl.timer_reset**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x4.pctl.timer_reset.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x4.pctl.timer_reset.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x4.pctl.timer_reset.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

# 13.34 FVP_Base_Cortex-A57x4-A35x4

List of instances in FVP_Base_Cortex-A57x4-A35x4.

## FVP_Base_Cortex-A57x4-A35x4 instances

**FVP_Base_Cortex_A57x4_A35x4**

Base Platform Compute Subsystem for ARMCortexA57x4CT and ARMCortexA35x4CT.
Type: `FVP_Base_Cortex_A57x4_A35x4`.

**FVP_Base_Cortex_A57x4_A35x4.bp**

Peripherals and address map for the Base Platform.
Type: `BasePlatformPeripherals`.

**FVP_Base_Cortex_A57x4_A35x4.bp.Timer_0_1**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Cortex_A57x4_A35x4.bp.Timer_0_1.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x4_A35x4.bp.Timer_0_1.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x4_A35x4.bp.Timer_0_1.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A57x4_A35x4.bp.Timer_0_1.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A57x4_A35x4.bp.Timer_2_3**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Cortex_A57x4_A35x4.bp.Timer_2_3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x4_A35x4.bp.Timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x4_A35x4.bp.Timer_2_3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A57x4_A35x4.bp.Timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A57x4_A35x4.bp.ap_refclk**

ARM Generic Timer.
Type: `MemoryMappedGenericTimer`.

**FVP_Base_Cortex_A57x4_A35x4.bp.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**FVP_Base_Cortex_A57x4_A35x4.bp.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x4_A35x4.bp.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x4_A35x4.bp.clock300MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x4_A35x4.bp.clock32KHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x4_A35x4.bp.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x4_A35x4.bp.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x4_A35x4.bp.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x4_A35x4.bp.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x4_A35x4.bp.dmc**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x4_A35x4.bp.dmc_phy**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x4_A35x4.bp.dummy_local_dap_rom**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x4_A35x4.bp.dummy_ram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x4_A35x4.bp.dummy_usb**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x4_A35x4.bp.exclusive_monitor**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x4_A35x4.bp.flash0**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A57x4_A35x4.bp.flash1**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A57x4_A35x4.bp.flashloader0**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A57x4_A35x4.bp.flashloader1**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A57x4_A35x4.bp.generic_watchdog**

ARM Generic Watchdog.
Type: `MemoryMappedGenericWatchdog`.

**FVP_Base_Cortex_A57x4_A35x4.bp.hdlcd0**

ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).
Type: `PL370_HDLCD`.

**FVP_Base_Cortex_A57x4_A35x4.bp.hdlcd0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x4_A35x4.bp.hdlcd0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x4_A35x4.bp.hdlcd0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x4_A35x4.bp.hdlcd0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x4_A35x4.bp.hdlcd0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A57x4_A35x4.bp.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_Base_Cortex_A57x4_A35x4.bp.mmc**

Generic Multimedia Card.
Type: `MMC`.

**FVP_Base_Cortex_A57x4_A35x4.bp.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x4_A35x4.bp.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x4_A35x4.bp.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x4_A35x4.bp.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x4_A35x4.bp.mpe**

Encrypt memory transactions for each encryption context with an independent key to prevent mismatch access.
Type: `PVMemoryProtectionEngine`.

**FVP_Base_Cortex_A57x4_A35x4.bp.nontrustedrom**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A57x4_A35x4.bp.nontrustedromloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x4_A35x4.bp.ns_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x4_A35x4.bp.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A57x4_A35x4.bp.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x4_A35x4.bp.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x4_A35x4.bp.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x4_A35x4.bp.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x4_A35x4.bp.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x4_A35x4.bp.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A57x4_A35x4.bp.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x4_A35x4.bp.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x4_A35x4.bp.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x4_A35x4.bp.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x4_A35x4.bp.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x4_A35x4.bp.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A57x4_A35x4.bp.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x4_A35x4.bp.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x4_A35x4.bp.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x4_A35x4.bp.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x4_A35x4.bp.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x4_A35x4.bp.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A57x4_A35x4.bp.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x4_A35x4.bp.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x4_A35x4.bp.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x4_A35x4.bp.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x4_A35x4.bp.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x4_A35x4.bp.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

**FVP_Base_Cortex_A57x4_A35x4.bp.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x4_A35x4.bp.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x4_A35x4.bp.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x4_A35x4.bp.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x4_A35x4.bp.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_Base_Cortex_A57x4_A35x4.bp.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x4_A35x4.bp.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x4_A35x4.bp.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x4_A35x4.bp.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x4_A35x4.bp.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A57x4_A35x4.bp.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x4_A35x4.bp.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A57x4_A35x4.bp.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x4_A35x4.bp.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_Base_Cortex_A57x4_A35x4.bp.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x4_A35x4.bp.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x4_A35x4.bp.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x4_A35x4.bp.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x4_A35x4.bp.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x4_A35x4.bp.pl111_clcd_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A57x4_A35x4.bp.pl180_mci**

ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

**FVP_Base_Cortex_A57x4_A35x4.bp.ps2keyboard**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

**FVP_Base_Cortex_A57x4_A35x4.bp.ps2keyboard.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x4_A35x4.bp.ps2keyboard.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x4_A35x4.bp.ps2keyboard.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x4_A35x4.bp.ps2keyboard.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x4_A35x4.bp.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

**FVP_Base_Cortex_A57x4_A35x4.bp.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x4_A35x4.bp.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x4_A35x4.bp.ps2mouse.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x4_A35x4.bp.ps2mouse.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x4_A35x4.bp.psram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x4_A35x4.bp.refcounter**
Memory Mapped Counter Module for Generic Timers.
Type: `MemoryMappedCounterModule`.

**FVP_Base_Cortex_A57x4_A35x4.bp.reset_or**
Or Gate.
Type: `OrGate`.

**FVP_Base_Cortex_A57x4_A35x4.bp.rl_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x4_A35x4.bp.rt_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x4_A35x4.bp.s_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x4_A35x4.bp.secureDRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x4_A35x4.bp.secureSRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x4_A35x4.bp.secureflash**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A57x4_A35x4.bp.secureflashloader**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A57x4_A35x4.bp.smsc_91c111**
SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_Base_Cortex_A57x4_A35x4.bp.sp805_wdog**
ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A57x4_A35x4.bp.sp805_wdog.clocktimer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x4_A35x4.bp.sp805_wdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x4_A35x4.bp.sp805_wdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x4_A35x4.bp.sp805_wdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x4_A35x4.bp.sp810_sysctrl**

Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_Base_Cortex_A57x4_A35x4.bp.sp810_sysctrl.clkdiv_clk0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x4_A35x4.bp.sp810_sysctrl.clkdiv_clk1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x4_A35x4.bp.sp810_sysctrl.clkdiv_clk2**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x4_A35x4.bp.sp810_sysctrl.clkdiv_clk3**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x4_A35x4.bp.sram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x4_A35x4.bp.terminal_0**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A57x4_A35x4.bp.terminal_1**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A57x4_A35x4.bp.terminal_2**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A57x4_A35x4.bp.terminal_3**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A57x4_A35x4.bp.trusted_key_storage**
Trusted Root-Key Storage unit.
Type: `RootKeyStorage`.

**FVP_Base_Cortex_A57x4_A35x4.bp.trusted_nv_counter**
Trusted Non-Volatile Counter unit.
Type: `NonVolatileCounter`.

**FVP_Base_Cortex_A57x4_A35x4.bp.trusted_rng**
Random Number Generator unit.
Type: `RandomNumberGenerator`.

**FVP_Base_Cortex_A57x4_A35x4.bp.trusted_watchdog**
ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A57x4_A35x4.bp.trusted_watchdog.clocktimer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x4_A35x4.bp.trusted_watchdog.clocktimer.timer**
A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x4_A35x4.bp.trusted_watchdog.clocktimer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x4_A35x4.bp.trusted_watchdog.clocktimer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x4_A35x4.bp.tzc_400**
TrustZone Address Space Controller.
Type: `TZC_400`.

**FVP_Base_Cortex_A57x4_A35x4.bp.ve_sysregs**
Type: `VE_SysRegs`.

**FVP_Base_Cortex_A57x4_A35x4.bp.vedcc**
Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

**FVP_Base_Cortex_A57x4_A35x4.bp.virtio_net**
VirtioNet device over MMIO transport.
Type: `VirtioNetMMIO`.

**FVP_Base_Cortex_A57x4_A35x4.bp.virtio_net_labeller**
Type: `Labeller`.

**FVP_Base_Cortex_A57x4_A35x4.bp.virtio_rng**
VirtioEntropy device over MMIO transport.
Type: `VirtioEntropyMMIO`.

**FVP_Base_Cortex_A57x4_A35x4.bp.virtioblockdevice**
VirtioBlock device over MMIO transport.
Type: `VirtioBlockMMIO`.

**FVP_Base_Cortex_A57x4_A35x4.bp.virtioblockdevice_labeller**
Type: `Labeller`.

**FVP_Base_Cortex_A57x4_A35x4.bp.virtiop9device**
virtio P9 server.
Type: `VirtioP9Device`.

**FVP_Base_Cortex_A57x4_A35x4.bp.virtiop9device_labeller**
Type: `Labeller`.

**FVP_Base_Cortex_A57x4_A35x4.bp.vis**
Display window for VE using Visualisation library.
Type: `VEVisualisation`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x4_A35x4.bp.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_Base_Cortex_A57x4_A35x4.bp.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x4_A35x4.bp.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x4_A35x4.bp.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x4_A35x4.bp.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x4_A35x4.bp.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x4_A35x4.bp.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x4_A35x4.bp.vram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x4_A35x4.cci400**

Cache Coherent Interconnect for AXI4 ACE.
Type: `CCI400`.

**FVP_Base_Cortex_A57x4_A35x4.clockdivider0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x4_A35x4.clockdivider1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x4_A35x4.cluster0**

ARM Cortex-A57 Cluster CT model.
Type: `Cluster_ARM_Cortex-A57`.

**FVP_Base_Cortex_A57x4_A35x4.cluster0.cpu0**

ARM Cortex-A57 CT model.
Type: `ARM_Cortex-A57`.

**FVP_Base_Cortex_A57x4_A35x4.cluster0.cpu0.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A57x4_A35x4.cluster0.cpu0.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A57x4_A35x4.cluster0.cpu0.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A57x4_A35x4.cluster0.cpu1**

ARM Cortex-A57 CT model.
Type: `ARM_Cortex-A57`.

**FVP_Base_Cortex_A57x4_A35x4.cluster0.cpu1.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A57x4_A35x4.cluster0.cpu1.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A57x4_A35x4.cluster0.cpu1.l1icache**

PV Cache.
Type: `PVCache`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x4_A35x4.cluster0.cpu2**

ARM Cortex-A57 CT model.
Type: `ARM_Cortex-A57`.

**FVP_Base_Cortex_A57x4_A35x4.cluster0.cpu2.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A57x4_A35x4.cluster0.cpu2.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A57x4_A35x4.cluster0.cpu2.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A57x4_A35x4.cluster0.cpu3**

ARM Cortex-A57 CT model.
Type: `ARM_Cortex-A57`.

**FVP_Base_Cortex_A57x4_A35x4.cluster0.cpu3.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A57x4_A35x4.cluster0.cpu3.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A57x4_A35x4.cluster0.cpu3.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A57x4_A35x4.cluster0.l2_cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A57x4_A35x4.cluster0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A57x4_A35x4.cluster1**

ARM Cortex-A35 Cluster CT model.
Type: `Cluster_ARM_Cortex-A35`.

**FVP_Base_Cortex_A57x4_A35x4.cluster1.cpu0**

ARM Cortex-A35 CT model.
Type: `ARM_Cortex-A35`.

**FVP_Base_Cortex_A57x4_A35x4.cluster1.cpu0.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A57x4_A35x4.cluster1.cpu0.l1dcache**

PV Cache.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `PVCache`.

**FVP_Base_Cortex_A57x4_A35x4.cluster1.cpu0.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A57x4_A35x4.cluster1.cpu1**

ARM Cortex-A35 CT model.
Type: `ARM_Cortex-A35`.

**FVP_Base_Cortex_A57x4_A35x4.cluster1.cpu1.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A57x4_A35x4.cluster1.cpu1.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A57x4_A35x4.cluster1.cpu1.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A57x4_A35x4.cluster1.cpu2**

ARM Cortex-A35 CT model.
Type: `ARM_Cortex-A35`.

**FVP_Base_Cortex_A57x4_A35x4.cluster1.cpu2.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A57x4_A35x4.cluster1.cpu2.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A57x4_A35x4.cluster1.cpu2.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A57x4_A35x4.cluster1.cpu3**

ARM Cortex-A35 CT model.
Type: `ARM_Cortex-A35`.

**FVP_Base_Cortex_A57x4_A35x4.cluster1.cpu3.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A57x4_A35x4.cluster1.cpu3.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A57x4_A35x4.cluster1.cpu3.l1icache**

PV Cache.
Type: `PVCache`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x4_A35x4.cluster1.l2_cache**

PV Cache.
Type: PVCache.

**FVP_Base_Cortex_A57x4_A35x4.cluster1_labeller**

Type: Labeller.

**FVP_Base_Cortex_A57x4_A35x4.dapmemlogger**

Bus Logger.
Type: PVBusLogger.

**FVP_Base_Cortex_A57x4_A35x4.elfloader**

ELF loader component.
Type: ElfLoader.

**FVP_Base_Cortex_A57x4_A35x4.gic_distributor**

GIC Interrupt Redistribution Infrastucture component.
Type: GIC_IRI.

**FVP_Base_Cortex_A57x4_A35x4.pctl**

Base Platforms Power Controller.
Type: Base_PowerController.

**FVP_Base_Cortex_A57x4_A35x4.pctl.timer_reset**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: ClockTimerThread.

**FVP_Base_Cortex_A57x4_A35x4.pctl.timer_reset.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: ClockTimerThread64.

**FVP_Base_Cortex_A57x4_A35x4.pctl.timer_reset.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: SchedulerThread.

**FVP_Base_Cortex_A57x4_A35x4.pctl.timer_reset.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: SchedulerThreadEvent.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

# 13.35 FVP_Base_Cortex-A57x4-A53x4

List of instances in FVP_Base_Cortex-A57x4-A53x4.

## FVP_Base_Cortex-A57x4-A53x4 instances

**FVP_Base_Cortex_A57x4_A53x4**

Base Platform Compute Subsystem for ARMCortexA57x4CT and ARMCortexA53x4CT.
Type: `FVP_Base_Cortex_A57x4_A53x4`.

**FVP_Base_Cortex_A57x4_A53x4.bp**

Peripherals and address map for the Base Platform.
Type: `BasePlatformPeripherals`.

**FVP_Base_Cortex_A57x4_A53x4.bp.Timer_0_1**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Cortex_A57x4_A53x4.bp.Timer_0_1.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x4_A53x4.bp.Timer_0_1.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x4_A53x4.bp.Timer_0_1.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A57x4_A53x4.bp.Timer_0_1.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A57x4_A53x4.bp.Timer_2_3**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Cortex_A57x4_A53x4.bp.Timer_2_3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x4_A53x4.bp.Timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x4_A53x4.bp.Timer_2_3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A57x4_A53x4.bp.Timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A57x4_A53x4.bp.ap_refclk**

ARM Generic Timer.
Type: `MemoryMappedGenericTimer`.

**FVP_Base_Cortex_A57x4_A53x4.bp.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**FVP_Base_Cortex_A57x4_A53x4.bp.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x4_A53x4.bp.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x4_A53x4.bp.clock300MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x4_A53x4.bp.clock32KHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x4_A53x4.bp.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x4_A53x4.bp.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x4_A53x4.bp.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x4_A53x4.bp.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x4_A53x4.bp.dmc**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x4_A53x4.bp.dmc_phy**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x4_A53x4.bp.dummy_local_dap_rom**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x4_A53x4.bp.dummy_ram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x4_A53x4.bp.dummy_usb**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x4_A53x4.bp.exclusive_monitor**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x4_A53x4.bp.flash0**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A57x4_A53x4.bp.flash1**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A57x4_A53x4.bp.flashloader0**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A57x4_A53x4.bp.flashloader1**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A57x4_A53x4.bp.generic_watchdog**

ARM Generic Watchdog.
Type: `MemoryMappedGenericWatchdog`.

**FVP_Base_Cortex_A57x4_A53x4.bp.hdlcd0**

ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).
Type: `PL370_HDLCD`.

**FVP_Base_Cortex_A57x4_A53x4.bp.hdlcd0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x4_A53x4.bp.hdlcd0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x4_A53x4.bp.hdlcd0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x4_A53x4.bp.hdlcd0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x4_A53x4.bp.hdlcd0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A57x4_A53x4.bp.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_Base_Cortex_A57x4_A53x4.bp.mmc**

Generic Multimedia Card.
Type: `MMC`.

**FVP_Base_Cortex_A57x4_A53x4.bp.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x4_A53x4.bp.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x4_A53x4.bp.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x4_A53x4.bp.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x4_A53x4.bp.mpe**

Encrypt memory transactions for each encryption context with an independent key to prevent mismatch access.
Type: `PVMemoryProtectionEngine`.

**FVP_Base_Cortex_A57x4_A53x4.bp.nontrustedrom**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A57x4_A53x4.bp.nontrustedromloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x4_A53x4.bp.ns_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x4_A53x4.bp.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A57x4_A53x4.bp.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x4_A53x4.bp.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x4_A53x4.bp.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x4_A53x4.bp.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x4_A53x4.bp.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x4_A53x4.bp.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A57x4_A53x4.bp.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x4_A53x4.bp.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x4_A53x4.bp.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x4_A53x4.bp.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x4_A53x4.bp.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x4_A53x4.bp.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A57x4_A53x4.bp.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x4_A53x4.bp.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x4_A53x4.bp.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x4_A53x4.bp.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x4_A53x4.bp.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x4_A53x4.bp.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A57x4_A53x4.bp.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x4_A53x4.bp.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x4_A53x4.bp.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x4_A53x4.bp.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x4_A53x4.bp.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x4_A53x4.bp.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

**FVP_Base_Cortex_A57x4_A53x4.bp.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x4_A53x4.bp.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x4_A53x4.bp.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x4_A53x4.bp.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x4_A53x4.bp.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_Base_Cortex_A57x4_A53x4.bp.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x4_A53x4.bp.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x4_A53x4.bp.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x4_A53x4.bp.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x4_A53x4.bp.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A57x4_A53x4.bp.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x4_A53x4.bp.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A57x4_A53x4.bp.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x4_A53x4.bp.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_Base_Cortex_A57x4_A53x4.bp.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x4_A53x4.bp.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x4_A53x4.bp.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x4_A53x4.bp.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x4_A53x4.bp.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x4_A53x4.bp.pl111_clcd_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A57x4_A53x4.bp.pl180_mci**

ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

**FVP_Base_Cortex_A57x4_A53x4.bp.ps2keyboard**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

**FVP_Base_Cortex_A57x4_A53x4.bp.ps2keyboard.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x4_A53x4.bp.ps2keyboard.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x4_A53x4.bp.ps2keyboard.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x4_A53x4.bp.ps2keyboard.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x4_A53x4.bp.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

**FVP_Base_Cortex_A57x4_A53x4.bp.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x4_A53x4.bp.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x4_A53x4.bp.ps2mouse.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x4_A53x4.bp.ps2mouse.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x4_A53x4.bp.psram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x4_A53x4.bp.refcounter**
Memory Mapped Counter Module for Generic Timers.
Type: `MemoryMappedCounterModule`.

**FVP_Base_Cortex_A57x4_A53x4.bp.reset_or**
Or Gate.
Type: `OrGate`.

**FVP_Base_Cortex_A57x4_A53x4.bp.rl_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x4_A53x4.bp.rt_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x4_A53x4.bp.s_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x4_A53x4.bp.secureDRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x4_A53x4.bp.secureSRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x4_A53x4.bp.secureflash**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A57x4_A53x4.bp.secureflashloader**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A57x4_A53x4.bp.smsc_91c111**
SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_Base_Cortex_A57x4_A53x4.bp.sp805_wdog**
ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A57x4_A53x4.bp.sp805_wdog.clocktimer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x4_A53x4.bp.sp805_wdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x4_A53x4.bp.sp805_wdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x4_A53x4.bp.sp805_wdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x4_A53x4.bp.sp810_sysctrl**

Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_Base_Cortex_A57x4_A53x4.bp.sp810_sysctrl.clkdiv_clk0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x4_A53x4.bp.sp810_sysctrl.clkdiv_clk1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x4_A53x4.bp.sp810_sysctrl.clkdiv_clk2**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x4_A53x4.bp.sp810_sysctrl.clkdiv_clk3**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x4_A53x4.bp.sram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A57x4_A53x4.bp.terminal_0**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A57x4_A53x4.bp.terminal_1**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A57x4_A53x4.bp.terminal_2**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A57x4_A53x4.bp.terminal_3**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A57x4_A53x4.bp.trusted_key_storage**
Trusted Root-Key Storage unit.
Type: `RootKeyStorage`.

**FVP_Base_Cortex_A57x4_A53x4.bp.trusted_nv_counter**
Trusted Non-Volatile Counter unit.
Type: `NonVolatileCounter`.

**FVP_Base_Cortex_A57x4_A53x4.bp.trusted_rng**
Random Number Generator unit.
Type: `RandomNumberGenerator`.

**FVP_Base_Cortex_A57x4_A53x4.bp.trusted_watchdog**
ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A57x4_A53x4.bp.trusted_watchdog.clocktimer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x4_A53x4.bp.trusted_watchdog.clocktimer.timer**
A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x4_A53x4.bp.trusted_watchdog.clocktimer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x4_A53x4.bp.trusted_watchdog.clocktimer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x4_A53x4.bp.tzc_400**
TrustZone Address Space Controller.
Type: `TZC_400`.

**FVP_Base_Cortex_A57x4_A53x4.bp.ve_sysregs**
Type: `VE_SysRegs`.

**FVP_Base_Cortex_A57x4_A53x4.bp.vedcc**
Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

**FVP_Base_Cortex_A57x4_A53x4.bp.virtio_net**
VirtioNet device over MMIO transport.
Type: `VirtioNetMMIO`.

**FVP_Base_Cortex_A57x4_A53x4.bp.virtio_net_labeller**
Type: `Labeller`.

**FVP_Base_Cortex_A57x4_A53x4.bp.virtio_rng**
VirtioEntropy device over MMIO transport.
Type: `VirtioEntropyMMIO`.

**FVP_Base_Cortex_A57x4_A53x4.bp.virtioblockdevice**
VirtioBlock device over MMIO transport.
Type: `VirtioBlockMMIO`.

**FVP_Base_Cortex_A57x4_A53x4.bp.virtioblockdevice_labeller**
Type: `Labeller`.

**FVP_Base_Cortex_A57x4_A53x4.bp.virtiop9device**
virtio P9 server.
Type: `VirtioP9Device`.

**FVP_Base_Cortex_A57x4_A53x4.bp.virtiop9device_labeller**
Type: `Labeller`.

**FVP_Base_Cortex_A57x4_A53x4.bp.vis**
Display window for VE using Visualisation library.
Type: `VEVisualisation`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x4_A53x4.bp.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_Base_Cortex_A57x4_A53x4.bp.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x4_A53x4.bp.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x4_A53x4.bp.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x4_A53x4.bp.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x4_A53x4.bp.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A57x4_A53x4.bp.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x4_A53x4.bp.vram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x4_A53x4.cci400**

Cache Coherent Interconnect for AXI4 ACE.
Type: `CCI400`.

**FVP_Base_Cortex_A57x4_A53x4.clockdivider0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x4_A53x4.clockdivider1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A57x4_A53x4.cluster0**

ARM Cortex-A57 Cluster CT model.
Type: `Cluster_ARM_Cortex-A57`.

**FVP_Base_Cortex_A57x4_A53x4.cluster0.cpu0**

ARM Cortex-A57 CT model.
Type: `ARM_Cortex-A57`.

**FVP_Base_Cortex_A57x4_A53x4.cluster0.cpu0.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A57x4_A53x4.cluster0.cpu0.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A57x4_A53x4.cluster0.cpu0.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A57x4_A53x4.cluster0.cpu1**

ARM Cortex-A57 CT model.
Type: `ARM_Cortex-A57`.

**FVP_Base_Cortex_A57x4_A53x4.cluster0.cpu1.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A57x4_A53x4.cluster0.cpu1.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A57x4_A53x4.cluster0.cpu1.l1icache**

PV Cache.
Type: `PVCache`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x4_A53x4.cluster0.cpu2**

ARM Cortex-A57 CT model.
Type: `ARM_Cortex-A57`.

**FVP_Base_Cortex_A57x4_A53x4.cluster0.cpu2.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A57x4_A53x4.cluster0.cpu2.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A57x4_A53x4.cluster0.cpu2.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A57x4_A53x4.cluster0.cpu3**

ARM Cortex-A57 CT model.
Type: `ARM_Cortex-A57`.

**FVP_Base_Cortex_A57x4_A53x4.cluster0.cpu3.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A57x4_A53x4.cluster0.cpu3.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A57x4_A53x4.cluster0.cpu3.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A57x4_A53x4.cluster0.l2_cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A57x4_A53x4.cluster0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A57x4_A53x4.cluster1**

ARM Cortex-A53 Cluster CT model.
Type: `Cluster_ARM_Cortex-A53`.

**FVP_Base_Cortex_A57x4_A53x4.cluster1.cpu0**

ARM Cortex-A53 CT model.
Type: `ARM_Cortex-A53`.

**FVP_Base_Cortex_A57x4_A53x4.cluster1.cpu0.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A57x4_A53x4.cluster1.cpu0.l1dcache**

PV Cache.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `PVCache`.

**FVP_Base_Cortex_A57x4_A53x4.cluster1.cpu0.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A57x4_A53x4.cluster1.cpu1**

ARM Cortex-A53 CT model.
Type: `ARM_Cortex-A53`.

**FVP_Base_Cortex_A57x4_A53x4.cluster1.cpu1.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A57x4_A53x4.cluster1.cpu1.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A57x4_A53x4.cluster1.cpu1.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A57x4_A53x4.cluster1.cpu2**

ARM Cortex-A53 CT model.
Type: `ARM_Cortex-A53`.

**FVP_Base_Cortex_A57x4_A53x4.cluster1.cpu2.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A57x4_A53x4.cluster1.cpu2.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A57x4_A53x4.cluster1.cpu2.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A57x4_A53x4.cluster1.cpu3**

ARM Cortex-A53 CT model.
Type: `ARM_Cortex-A53`.

**FVP_Base_Cortex_A57x4_A53x4.cluster1.cpu3.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A57x4_A53x4.cluster1.cpu3.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A57x4_A53x4.cluster1.cpu3.l1icache**

PV Cache.
Type: `PVCache`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A57x4_A53x4.cluster1.l2_cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A57x4_A53x4.cluster1_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A57x4_A53x4.dapmemlogger**

Bus Logger.
Type: `PVBusLogger`.

**FVP_Base_Cortex_A57x4_A53x4.elfloader**

ELF loader component.
Type: `ElfLoader`.

**FVP_Base_Cortex_A57x4_A53x4.gic_distributor**

GIC Interrupt Redistribution Infrastucture component.
Type: `GIC_IRI`.

**FVP_Base_Cortex_A57x4_A53x4.pctl**

Base Platforms Power Controller.
Type: `Base_PowerController`.

**FVP_Base_Cortex_A57x4_A53x4.pctl.timer_reset**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A57x4_A53x4.pctl.timer_reset.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A57x4_A53x4.pctl.timer_reset.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A57x4_A53x4.pctl.timer_reset.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

# 13.36 FVP_Base_Cortex-A65

List of instances in FVP_Base_Cortex-A65.

## FVP_Base_Cortex-A65 instances

**FVP_Base_Cortex_A65**

Base Platform Compute Subsystem for ARMCortexA65CT.
Type: `FVP_Base_Cortex_A65`.

**FVP_Base_Cortex_A65.bp**

Peripherals and address map for the Base Platform.
Type: `BasePlatformPeripherals`.

**FVP_Base_Cortex_A65.bp.Timer_0_1**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Cortex_A65.bp.Timer_0_1.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65.bp.Timer_0_1.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65.bp.Timer_0_1.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A65.bp.Timer_0_1.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A65.bp.Timer_2_3**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Cortex_A65.bp.Timer_2_3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A65.bp.Timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65.bp.Timer_2_3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A65.bp.Timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A65.bp.ap_refclk**

ARM Generic Timer.
Type: `MemoryMappedGenericTimer`.

**FVP_Base_Cortex_A65.bp.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**FVP_Base_Cortex_A65.bp.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65.bp.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65.bp.clock300MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65.bp.clock32KHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A65.bp.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65.bp.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65.bp.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65.bp.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65.bp.dmc**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A65.bp.dmc_phy**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A65.bp.dummy_local_dap_rom**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A65.bp.dummy_ram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A65.bp.dummy_usb**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A65.bp.exclusive_monitor**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A65.bp.flash0**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A65.bp.flash1**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A65.bp.flashloader0**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A65.bp.flashloader1**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A65.bp.generic_watchdog**

ARM Generic Watchdog.
Type: `MemoryMappedGenericWatchdog`.

**FVP_Base_Cortex_A65.bp.hdlcd0**

ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).
Type: `PL370_HDLCD`.

**FVP_Base_Cortex_A65.bp.hdlcd0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A65.bp.hdlcd0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A65.bp.hdlcd0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A65.bp.hdlcd0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A65.bp.hdlcd0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A65.bp.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_Base_Cortex_A65.bp.mmc**

Generic Multimedia Card.
Type: `MMC`.

**FVP_Base_Cortex_A65.bp.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A65.bp.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A65.bp.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A65.bp.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A65.bp.mpe**

Encrypt memory transactions for each encryption context with an independent key to prevent mismatch access.
Type: `PVMemoryProtectionEngine`.

**FVP_Base_Cortex_A65.bp.nontrustedrom**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A65.bp.nontrustedromloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A65.bp.ns_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A65.bp.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A65.bp.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65.bp.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A65.bp.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A65.bp.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A65.bp.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A65.bp.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A65.bp.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65.bp.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A65.bp.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A65.bp.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A65.bp.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A65.bp.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A65.bp.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65.bp.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ClockTimerThread`.

**FVP_Base_Cortex_A65.bp.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A65.bp.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A65.bp.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A65.bp.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A65.bp.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65.bp.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A65.bp.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A65.bp.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A65.bp.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A65.bp.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

**FVP_Base_Cortex_A65.bp.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A65.bp.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A65.bp.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A65.bp.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A65.bp.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_Base_Cortex_A65.bp.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A65.bp.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A65.bp.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A65.bp.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A65.bp.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A65.bp.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65.bp.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A65.bp.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65.bp.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_Base_Cortex_A65.bp.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A65.bp.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A65.bp.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A65.bp.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A65.bp.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A65.bp.pl111_clcd_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A65.bp.pl180_mci**

ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

**FVP_Base_Cortex_A65.bp.ps2keyboard**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

**FVP_Base_Cortex_A65.bp.ps2keyboard.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A65.bp.ps2keyboard.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A65.bp.ps2keyboard.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A65.bp.ps2keyboard.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A65.bp.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

**FVP_Base_Cortex_A65.bp.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A65.bp.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A65.bp.ps2mouse.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A65.bp.ps2mouse.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A65.bp.psram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A65.bp.refcounter**
Memory Mapped Counter Module for Generic Timers.
Type: `MemoryMappedCounterModule`.

**FVP_Base_Cortex_A65.bp.reset_or**
Or Gate.
Type: `OrGate`.

**FVP_Base_Cortex_A65.bp.rl_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A65.bp.rt_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A65.bp.s_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A65.bp.secureDRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A65.bp.secureSRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A65.bp.secureflash**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A65.bp.secureflashloader**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A65.bp.smsc_91c111**
SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_Base_Cortex_A65.bp.sp805_wdog**
ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A65.bp.sp805_wdog.clocktimer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A65.bp.sp805_wdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A65.bp.sp805_wdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A65.bp.sp805_wdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A65.bp.sp810_sysctrl**

Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_Base_Cortex_A65.bp.sp810_sysctrl.clkdiv_clk0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65.bp.sp810_sysctrl.clkdiv_clk1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65.bp.sp810_sysctrl.clkdiv_clk2**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65.bp.sp810_sysctrl.clkdiv_clk3**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65.bp.sram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A65.bp.terminal_0**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A65.bp.terminal_1**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A65.bp.terminal_2**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A65.bp.terminal_3**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A65.bp.trusted_key_storage**
Trusted Root-Key Storage unit.
Type: `RootKeyStorage`.

**FVP_Base_Cortex_A65.bp.trusted_nv_counter**
Trusted Non-Volatile Counter unit.
Type: `NonVolatileCounter`.

**FVP_Base_Cortex_A65.bp.trusted_rng**
Random Number Generator unit.
Type: `RandomNumberGenerator`.

**FVP_Base_Cortex_A65.bp.trusted_watchdog**
ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A65.bp.trusted_watchdog.clocktimer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A65.bp.trusted_watchdog.clocktimer.timer**
A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A65.bp.trusted_watchdog.clocktimer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A65.bp.trusted_watchdog.clocktimer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A65.bp.tzc_400**
TrustZone Address Space Controller.
Type: `TZC_400`.

**FVP_Base_Cortex_A65.bp.ve_sysregs**
Type: `VE_SysRegs`.

**FVP_Base_Cortex_A65.bp.vedcc**
Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

**FVP_Base_Cortex_A65.bp.virtio_net**
VirtioNet device over MMIO transport.
Type: `VirtioNetMMIO`.

**FVP_Base_Cortex_A65.bp.virtio_net_labeller**
Type: `Labeller`.

**FVP_Base_Cortex_A65.bp.virtio_rng**
VirtioEntropy device over MMIO transport.
Type: `VirtioEntropyMMIO`.

**FVP_Base_Cortex_A65.bp.virtioblockdevice**
VirtioBlock device over MMIO transport.
Type: `VirtioBlockMMIO`.

**FVP_Base_Cortex_A65.bp.virtioblockdevice_labeller**
Type: `Labeller`.

**FVP_Base_Cortex_A65.bp.virtiop9device**
virtio P9 server.
Type: `VirtioP9Device`.

**FVP_Base_Cortex_A65.bp.virtiop9device_labeller**
Type: `Labeller`.

**FVP_Base_Cortex_A65.bp.vis**
Display window for VE using Visualisation library.
Type: `VEVisualisation`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A65.bp.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_Base_Cortex_A65.bp.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65.bp.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A65.bp.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A65.bp.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A65.bp.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A65.bp.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65.bp.vram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A65.cci400**

Cache Coherent Interconnect for AXI4 ACE.
Type: `CCI400`.

**FVP_Base_Cortex_A65.clockdivider0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65.cluster0**

ARM Cortex-A65 Cluster CT model.
Type: `Cluster_ARM_Cortex-A65`.

**FVP_Base_Cortex_A65.cluster0.DSU.shared_cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A65.cluster0.cpu0.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A65.cluster0.cpu0.thread0**

ARM Cortex-A65 CT model.
Type: `ARM_Cortex-A65`.

**FVP_Base_Cortex_A65.cluster0.cpu0.thread1**

ARM Cortex-A65 CT model.
Type: `ARM_Cortex-A65`.

**FVP_Base_Cortex_A65.cluster0.cpu1.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A65.cluster0.cpu1.thread0**

ARM Cortex-A65 CT model.
Type: `ARM_Cortex-A65`.

**FVP_Base_Cortex_A65.cluster0.cpu1.thread1**

ARM Cortex-A65 CT model.
Type: `ARM_Cortex-A65`.

**FVP_Base_Cortex_A65.cluster0.cpu2.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A65.cluster0.cpu2.thread0**

ARM Cortex-A65 CT model.
Type: `ARM_Cortex-A65`.

**FVP_Base_Cortex_A65.cluster0.cpu2.thread1**

ARM Cortex-A65 CT model.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ARM_Cortex-A65`.

**FVP_Base_Cortex_A65.cluster0.cpu3.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A65.cluster0.cpu3.thread0**

ARM Cortex-A65 CT model.
Type: `ARM_Cortex-A65`.

**FVP_Base_Cortex_A65.cluster0.cpu3.thread1**

ARM Cortex-A65 CT model.
Type: `ARM_Cortex-A65`.

**FVP_Base_Cortex_A65.cluster0.cpu4.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A65.cluster0.cpu4.thread0**

ARM Cortex-A65 CT model.
Type: `ARM_Cortex-A65`.

**FVP_Base_Cortex_A65.cluster0.cpu4.thread1**

ARM Cortex-A65 CT model.
Type: `ARM_Cortex-A65`.

**FVP_Base_Cortex_A65.cluster0.cpu5.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A65.cluster0.cpu5.thread0**

ARM Cortex-A65 CT model.
Type: `ARM_Cortex-A65`.

**FVP_Base_Cortex_A65.cluster0.cpu5.thread1**

ARM Cortex-A65 CT model.
Type: `ARM_Cortex-A65`.

**FVP_Base_Cortex_A65.cluster0.cpu6.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A65.cluster0.cpu6.thread0**

ARM Cortex-A65 CT model.
Type: `ARM_Cortex-A65`.

**FVP_Base_Cortex_A65.cluster0.cpu6.thread1**

ARM Cortex-A65 CT model.
Type: `ARM_Cortex-A65`.

**FVP_Base_Cortex_A65.cluster0.cpu7.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A65.cluster0.cpu7.thread0**

ARM Cortex-A65 CT model.
Type: `ARM_Cortex-A65`.

**FVP_Base_Cortex_A65.cluster0.cpu7.thread1**

ARM Cortex-A65 CT model.
Type: `ARM_Cortex-A65`.

**FVP_Base_Cortex_A65.cluster0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A65.dapmemlogger**

Bus Logger.
Type: `PVBusLogger`.

**FVP_Base_Cortex_A65.elfloader**

ELF loader component.
Type: `ElfLoader`.

**FVP_Base_Cortex_A65.gic_distributor**

GIC Interrupt Redistribution Infrastucture component.
Type: `GIC_IRI`.

**FVP_Base_Cortex_A65.pctl**

Base Platforms Power Controller.
Type: `Base_PowerController`.

**FVP_Base_Cortex_A65.pctl.timer_reset**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A65.pctl.timer_reset.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A65.pctl.timer_reset.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A65.pctl.timer_reset.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

# 13.37 FVP_Base_Cortex-A65AE

List of instances in FVP_Base_Cortex-A65AE.

## FVP_Base_Cortex-A65AE instances

**FVP_Base_Cortex_A65AE**

Base Platform Compute Subsystem for ARMCortexA65AECT.
Type: `FVP_Base_Cortex_A65AE`.

**FVP_Base_Cortex_A65AE.bp**

Peripherals and address map for the Base Platform.
Type: `BasePlatformPeripherals`.

**FVP_Base_Cortex_A65AE.bp.Timer_0_1**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Cortex_A65AE.bp.Timer_0_1.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65AE.bp.Timer_0_1.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65AE.bp.Timer_0_1.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A65AE.bp.Timer_0_1.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A65AE.bp.Timer_2_3**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A65AE.bp.Timer_2_3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65AE.bp.Timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65AE.bp.Timer_2_3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A65AE.bp.Timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A65AE.bp.ap_refclk**

ARM Generic Timer.
Type: `MemoryMappedGenericTimer`.

**FVP_Base_Cortex_A65AE.bp.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**FVP_Base_Cortex_A65AE.bp.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65AE.bp.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65AE.bp.clock300MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A65AE.bp.clock32KHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65AE.bp.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65AE.bp.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65AE.bp.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65AE.bp.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65AE.bp.dmc**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A65AE.bp.dmc_phy**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A65AE.bp.dummy_local_dap_rom**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A65AE.bp.dummy_ram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A65AE.bp.dummy_usb**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A65AE.bp.exclusive_monitor**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_Base_Cortex_A65AE.bp.flash0**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A65AE.bp.flash1**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A65AE.bp.flashloader0**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A65AE.bp.flashloader1**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A65AE.bp.generic_watchdog**

ARM Generic Watchdog.
Type: `MemoryMappedGenericWatchdog`.

**FVP_Base_Cortex_A65AE.bp.hdlcd0**

ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).
Type: `PL370_HDLCD`.

**FVP_Base_Cortex_A65AE.bp.hdlcd0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A65AE.bp.hdlcd0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A65AE.bp.hdlcd0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A65AE.bp.hdlcd0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A65AE.bp.hdlcd0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A65AE.bp.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_Base_Cortex_A65AE.bp.mmc**

Generic Multimedia Card.
Type: `MMC`.

**FVP_Base_Cortex_A65AE.bp.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A65AE.bp.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A65AE.bp.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A65AE.bp.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A65AE.bp.mpe**

Encrypt memory transactions for each encryption context with an independent key to prevent mismatch access.
Type: `PVMemoryProtectionEngine`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A65AE.bp.nontrustedrom**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A65AE.bp.nontrustedromloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A65AE.bp.ns_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A65AE.bp.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A65AE.bp.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65AE.bp.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A65AE.bp.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A65AE.bp.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A65AE.bp.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A65AE.bp.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A65AE.bp.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65AE.bp.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A65AE.bp.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A65AE.bp.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A65AE.bp.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A65AE.bp.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A65AE.bp.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A65AE.bp.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A65AE.bp.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A65AE.bp.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A65AE.bp.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A65AE.bp.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A65AE.bp.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65AE.bp.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A65AE.bp.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A65AE.bp.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A65AE.bp.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A65AE.bp.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

**FVP_Base_Cortex_A65AE.bp.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A65AE.bp.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A65AE.bp.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A65AE.bp.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A65AE.bp.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_Base_Cortex_A65AE.bp.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A65AE.bp.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A65AE.bp.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A65AE.bp.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A65AE.bp.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A65AE.bp.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65AE.bp.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A65AE.bp.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65AE.bp.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_Base_Cortex_A65AE.bp.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

**FVP_Base_Cortex_A65AE.bp.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A65AE.bp.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A65AE.bp.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A65AE.bp.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A65AE.bp.pl111_clcd_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A65AE.bp.pl180_mci**

ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

**FVP_Base_Cortex_A65AE.bp.ps2keyboard**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A65AE.bp.ps2keyboard.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A65AE.bp.ps2keyboard.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A65AE.bp.ps2keyboard.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A65AE.bp.ps2keyboard.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A65AE.bp.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

**FVP_Base_Cortex_A65AE.bp.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A65AE.bp.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A65AE.bp.ps2mouse.ps2_clocktimer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A65AE.bp.ps2mouse.ps2_clocktimer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A65AE.bp.psram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A65AE.bp.refcounter**
Memory Mapped Counter Module for Generic Timers.
Type: `MemoryMappedCounterModule`.

**FVP_Base_Cortex_A65AE.bp.reset_or**
Or Gate.
Type: `OrGate`.

**FVP_Base_Cortex_A65AE.bp.rl_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A65AE.bp.rt_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A65AE.bp.s_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A65AE.bp.secureDRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A65AE.bp.secureSRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A65AE.bp.secureflash**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A65AE.bp.secureflashloader**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A65AE.bp.smsc_91c111**

SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_Base_Cortex_A65AE.bp.sp805_wdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A65AE.bp.sp805_wdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A65AE.bp.sp805_wdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A65AE.bp.sp805_wdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A65AE.bp.sp805_wdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A65AE.bp.sp810_sysctrl**

Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_Base_Cortex_A65AE.bp.sp810_sysctrl.clkdiv_clk0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65AE.bp.sp810_sysctrl.clkdiv_clk1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65AE.bp.sp810_sysctrl.clkdiv_clk2**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65AE.bp.sp810_sysctrl.clkdiv_clk3**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65AE.bp.sram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A65AE.bp.terminal_0**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A65AE.bp.terminal_1**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A65AE.bp.terminal_2**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A65AE.bp.terminal_3**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A65AE.bp.trusted_key_storage**
Trusted Root-Key Storage unit.
Type: `RootKeyStorage`.

**FVP_Base_Cortex_A65AE.bp.trusted_nv_counter**
Trusted Non-Volatile Counter unit.
Type: `NonVolatileCounter`.

**FVP_Base_Cortex_A65AE.bp.trusted_rng**
Random Number Generator unit.
Type: `RandomNumberGenerator`.

**FVP_Base_Cortex_A65AE.bp.trusted_watchdog**
ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A65AE.bp.trusted_watchdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A65AE.bp.trusted_watchdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A65AE.bp.trusted_watchdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A65AE.bp.trusted_watchdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A65AE.bp.tzc_400**

TrustZone Address Space Controller.
Type: `TZC_400`.

**FVP_Base_Cortex_A65AE.bp.ve_sysregs**

Type: `VE_SysRegs`.

**FVP_Base_Cortex_A65AE.bp.vedcc**

Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

**FVP_Base_Cortex_A65AE.bp.virtio_net**

VirtioNet device over MMIO transport.
Type: `VirtioNetMMIO`.

**FVP_Base_Cortex_A65AE.bp.virtio_net_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A65AE.bp.virtio_rng**

VirtioEntropy device over MMIO transport.
Type: `VirtioEntropyMMIO`.

**FVP_Base_Cortex_A65AE.bp.virtioblockdevice**

VirtioBlock device over MMIO transport.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `VirtioBlockMMIO`.

**FVP_Base_Cortex_A65AE.bp.virtioblockdevice_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A65AE.bp.virtiop9device**

virtio P9 server.
Type: `VirtioP9Device`.

**FVP_Base_Cortex_A65AE.bp.virtiop9device_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A65AE.bp.vis**

Display window for VE using Visualisation library.
Type: `VEVisualisation`.

**FVP_Base_Cortex_A65AE.bp.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_Base_Cortex_A65AE.bp.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65AE.bp.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A65AE.bp.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A65AE.bp.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A65AE.bp.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A65AE.bp.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65AE.bp.vram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A65AE.cci400**

Cache Coherent Interconnect for AXI4 ACE.
Type: `CCI400`.

**FVP_Base_Cortex_A65AE.clockdivider0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65AE.cluster0**

ARM Cortex-A65AE Cluster CT model.
Type: `Cluster_ARM_Cortex-A65AE`.

**FVP_Base_Cortex_A65AE.cluster0.DSU.shared_cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A65AE.cluster0.cpu0.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A65AE.cluster0.cpu0.thread0**

ARM Cortex-A65AE CT model.
Type: `ARM_Cortex-A65AE`.

**FVP_Base_Cortex_A65AE.cluster0.cpu0.thread1**

ARM Cortex-A65AE CT model.
Type: `ARM_Cortex-A65AE`.

**FVP_Base_Cortex_A65AE.cluster0.cpu1.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A65AE.cluster0.cpu1.thread0**

ARM Cortex-A65AE CT model.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ARM_Cortex-A65AE`.

**FVP_Base_Cortex_A65AE.cluster0.cpu1.thread1**

ARM Cortex-A65AE CT model.
Type: `ARM_Cortex-A65AE`.

**FVP_Base_Cortex_A65AE.cluster0.cpu2.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A65AE.cluster0.cpu2.thread0**

ARM Cortex-A65AE CT model.
Type: `ARM_Cortex-A65AE`.

**FVP_Base_Cortex_A65AE.cluster0.cpu2.thread1**

ARM Cortex-A65AE CT model.
Type: `ARM_Cortex-A65AE`.

**FVP_Base_Cortex_A65AE.cluster0.cpu3.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A65AE.cluster0.cpu3.thread0**

ARM Cortex-A65AE CT model.
Type: `ARM_Cortex-A65AE`.

**FVP_Base_Cortex_A65AE.cluster0.cpu3.thread1**

ARM Cortex-A65AE CT model.
Type: `ARM_Cortex-A65AE`.

**FVP_Base_Cortex_A65AE.cluster0.cpu4.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A65AE.cluster0.cpu4.thread0**

ARM Cortex-A65AE CT model.
Type: `ARM_Cortex-A65AE`.

**FVP_Base_Cortex_A65AE.cluster0.cpu4.thread1**

ARM Cortex-A65AE CT model.
Type: `ARM_Cortex-A65AE`.

**FVP_Base_Cortex_A65AE.cluster0.cpu5.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A65AE.cluster0.cpu5.thread0**

ARM Cortex-A65AE CT model.
Type: `ARM_Cortex-A65AE`.

**FVP_Base_Cortex_A65AE.cluster0.cpu5.thread1**

ARM Cortex-A65AE CT model.
Type: `ARM_Cortex-A65AE`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A65AE.cluster0.cpu6.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A65AE.cluster0.cpu6.thread0**

ARM Cortex-A65AE CT model.
Type: `ARM_Cortex-A65AE`.

**FVP_Base_Cortex_A65AE.cluster0.cpu6.thread1**

ARM Cortex-A65AE CT model.
Type: `ARM_Cortex-A65AE`.

**FVP_Base_Cortex_A65AE.cluster0.cpu7.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A65AE.cluster0.cpu7.thread0**

ARM Cortex-A65AE CT model.
Type: `ARM_Cortex-A65AE`.

**FVP_Base_Cortex_A65AE.cluster0.cpu7.thread1**

ARM Cortex-A65AE CT model.
Type: `ARM_Cortex-A65AE`.

**FVP_Base_Cortex_A65AE.cluster0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A65AE.dapmemlogger**

Bus Logger.
Type: `PVBusLogger`.

**FVP_Base_Cortex_A65AE.elfloader**

ELF loader component.
Type: `ElfLoader`.

**FVP_Base_Cortex_A65AE.gic_distributor**

GIC Interrupt Redistribution Infrastucture component.
Type: `GIC_IRI`.

**FVP_Base_Cortex_A65AE.pctl**

Base Platforms Power Controller.
Type: `Base_PowerController`.

**FVP_Base_Cortex_A65AE.pctl.timer_reset**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A65AE.pctl.timer_reset.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A65AE.pctl.timer_reset.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A65AE.pctl.timer_reset.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

## 13.38 FVP_Base_Cortex-A65AE+Cortex-A76AE

List of instances in FVP_Base_Cortex-A65AE+Cortex-A76AE.

### FVP_Base_Cortex-A65AE+Cortex-A76AE instances

**FVP_Base_Cortex_A65AE_Cortex_A76AE**

Base Platform Compute Subsystem for ARMCortexA65AECT_CortexA76AECT.
Type: `FVP_Base_Cortex_A65AE_Cortex_A76AE`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp**

Peripherals and address map for the Base Platform.
Type: `BasePlatformPeripherals`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.Timer_0_1**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.Timer_0_1.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.Timer_0_1.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.Timer_0_1.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.Timer_0_1.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.Timer_2_3**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.Timer_2_3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.Timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.Timer_2_3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.Timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.ap_refclk**

ARM Generic Timer.
Type: `MemoryMappedGenericTimer`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.clock300MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.clock32KHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.dmc**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.dmc_phy**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.dummy_local_dap_rom**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.dummy_ram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.dummy_usb**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.exclusive_monitor**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.flash0**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.flash1**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.flashloader0**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.flashloader1**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.generic_watchdog**

ARM Generic Watchdog.
Type: `MemoryMappedGenericWatchdog`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.hdlcd0**

ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).
Type: `PL370_HDLCD`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.hdlcd0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.hdlcd0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.hdlcd0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.hdlcd0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.hdlcd0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.mmc**

Generic Multimedia Card.
Type: `MMC`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.mpe**

Encrypt memory transactions for each encryption context with an independent key to prevent mismatch access.
Type: `PVMemoryProtectionEngine`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.nontrustedrom**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.nontrustedromloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.ns_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.pl111_clcd_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.pl180_mci**

ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.ps2keyboard**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.ps2keyboard.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.ps2keyboard.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.ps2keyboard.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.ps2keyboard.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.ps2mouse.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.ps2mouse.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.psram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.refcounter**

Memory Mapped Counter Module for Generic Timers.
Type: `MemoryMappedCounterModule`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.reset_or**

Or Gate.
Type: `OrGate`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.rl_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.rt_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.s_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.secureDRAM**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.secureSRAM**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.secureflash**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.secureflashloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.smsc_91c111**

SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.sp805_wdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.sp805_wdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.sp805_wdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.sp805_wdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.sp805_wdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.sp810_sysctrl**

Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.sp810_sysctrl.clkdiv_clk0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.sp810_sysctrl.clkdiv_clk1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.sp810_sysctrl.clkdiv_clk2**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.sp810_sysctrl.clkdiv_clk3**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.sram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.terminal_0**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.terminal_1**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.terminal_2**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.terminal_3**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.trusted_key_storage**

Trusted Root-Key Storage unit.
Type: `RootKeyStorage`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.trusted_nv_counter**

Trusted Non-Volatile Counter unit.
Type: `NonVolatileCounter`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.trusted_rng**

Random Number Generator unit.
Type: `RandomNumberGenerator`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.trusted_watchdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.trusted_watchdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.trusted_watchdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.trusted_watchdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.trusted_watchdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.tzc_400**

TrustZone Address Space Controller.
Type: `TZC_400`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.ve_sysregs**

Type: `VE_SysRegs`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.vedcc**

Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.virtio_net**

VirtioNet device over MMIO transport.
Type: `VirtioNetMMIO`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.virtio_net_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.virtio_rng**

VirtioEntropy device over MMIO transport.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `VirtioEntropyMMIO`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.virtioblockdevice**

VirtioBlock device over MMIO transport.
Type: `VirtioBlockMMIO`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.virtioblockdevice_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.virtiop9device**

virtio P9 server.
Type: `VirtioP9Device`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.virtiop9device_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.vis**

Display window for VE using Visualisation library.
Type: `VEVisualisation`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.bp.vram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.cci400**

Cache Coherent Interconnect for AXI4 ACE.
Type: `CCI400`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.clockdivider0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.cluster0**

ARM Cortex-A65AE_Cortex-A76AE Cluster CT model.
Type: `Cluster_ARM_Cortex-A65AE_Cortex-A76AE`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.cluster0.DSU.shared_cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.cluster0.subcluster0**

ARM Cortex-A65AE Cluster CT model.
Type: `Subcluster_ARM_Cortex-A65AE`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.cluster0.subcluster0.cpu0.thread0**

ARM Cortex-A65AE CT model.
Type: `ARM_Cortex-A65AE`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.cluster0.subcluster0.cpu0.thread1**

ARM Cortex-A65AE CT model.
Type: `ARM_Cortex-A65AE`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.cluster0.subcluster0.cpu1.thread0**

ARM Cortex-A65AE CT model.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ARM_Cortex-A65AE`.

**`FVP_Base_Cortex_A65AE_Cortex_A76AE.cluster0.subcluster0.cpu1.thread1`**

ARM Cortex-A65AE CT model.
Type: `ARM_Cortex-A65AE`.

**`FVP_Base_Cortex_A65AE_Cortex_A76AE.cluster0.subcluster1`**

ARM Cortex-A76AE Cluster CT model.
Type: `Subcluster_ARM_Cortex-A76AE`.

**`FVP_Base_Cortex_A65AE_Cortex_A76AE.cluster0.subcluster1.cpu0`**

ARM Cortex-A76AE CT model.
Type: `ARM_Cortex-A76AE`.

**`FVP_Base_Cortex_A65AE_Cortex_A76AE.cluster0.subcluster1.cpu0.l1dcache`**

PV Cache.
Type: `PVCache`.

**`FVP_Base_Cortex_A65AE_Cortex_A76AE.cluster0.subcluster1.cpu0.l1icache`**

PV Cache.
Type: `PVCache`.

**`FVP_Base_Cortex_A65AE_Cortex_A76AE.cluster0.subcluster1.cpu0.l2cache`**

PV Cache.
Type: `PVCache`.

**`FVP_Base_Cortex_A65AE_Cortex_A76AE.cluster0.subcluster1.cpu1`**

ARM Cortex-A76AE CT model.
Type: `ARM_Cortex-A76AE`.

**`FVP_Base_Cortex_A65AE_Cortex_A76AE.cluster0.subcluster1.cpu1.l1dcache`**

PV Cache.
Type: `PVCache`.

**`FVP_Base_Cortex_A65AE_Cortex_A76AE.cluster0.subcluster1.cpu1.l1icache`**

PV Cache.
Type: `PVCache`.

**`FVP_Base_Cortex_A65AE_Cortex_A76AE.cluster0.subcluster1.cpu1.l2cache`**

PV Cache.
Type: `PVCache`.

**`FVP_Base_Cortex_A65AE_Cortex_A76AE.cluster0_labeller`**

Type: `Labeller`.

**`FVP_Base_Cortex_A65AE_Cortex_A76AE.dapmemlogger`**

Bus Logger.
Type: `PVBusLogger`.

**`FVP_Base_Cortex_A65AE_Cortex_A76AE.elfloader`**

ELF loader component.
Type: `ElfLoader`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A65AE_Cortex_A76AE.gic_distributor**

GIC Interrupt Redistribution Infrastucture component.
Type: `GIC_IRI`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.pctl**

Base Platforms Power Controller.
Type: `Base_PowerController`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.pctl.timer_reset**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.pctl.timer_reset.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.pctl.timer_reset.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A65AE_Cortex_A76AE.pctl.timer_reset.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

## 13.39 FVP_Base_Cortex-A65AEx2+Cortex-A76AEx2

List of instances in FVP_Base_Cortex-A65AEx2+Cortex-A76AEx2.

### FVP_Base_Cortex-A65AEx2+Cortex-A76AEx2 instances

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2**

Base Platform Compute Subsystem for ARMCortexA65AEx2CT_CortexA76AEx2CT.
Type: `FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp**

Peripherals and address map for the Base Platform.
Type: `BasePlatformPeripherals`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.Timer_0_1**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.Timer_0_1.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.Timer_0_1.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.Timer_0_1.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.Timer_0_1.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.Timer_2_3**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.Timer_2_3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.Timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.Timer_2_3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.Timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.ap_refclk**

ARM Generic Timer.
Type: `MemoryMappedGenericTimer`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.clock300MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.clock32KHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.dmc**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.dmc_phy**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.dummy_local_dap_rom**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.dummy_ram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.dummy_usb**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.exclusive_monitor**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.flash0**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.flash1**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.flashloader0**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.flashloader1**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.generic_watchdog**

ARM Generic Watchdog.
Type: `MemoryMappedGenericWatchdog`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.hdlcd0**

ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).
Type: `PL370_HDLCD`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.hdlcd0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.hdlcd0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.hdlcd0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.hdlcd0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.hdlcd0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.mmc**

Generic Multimedia Card.
Type: `MMC`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.mpe**

Encrypt memory transactions for each encryption context with an independent key to prevent mismatch access.
Type: `PVMemoryProtectionEngine`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.nontrustedrom**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.nontrustedromloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.ns_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal()

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.pl111_clcd.pl11x_clcd.timer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.pl111_clcd.pl11x_clcd.timer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.pl111_clcd_labeller**
Type: `Labeller`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.pl180_mci**
ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.ps2keyboard**
Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.ps2keyboard.ps2_clocktimer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.ps2keyboard.ps2_clocktimer.timer**
A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.ps2keyboard.ps2_clocktimer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.ps2keyboard.ps2_clocktimer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.ps2mouse.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.ps2mouse.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.psram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.refcounter**

Memory Mapped Counter Module for Generic Timers.
Type: `MemoryMappedCounterModule`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.reset_or**

Or Gate.
Type: `OrGate`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.rl_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.rt_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.s_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.secureDRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.secureSRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.secureflash**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.secureflashloader**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.smsc_91c111**
SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.sp805_wdog**
ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.sp805_wdog.clocktimer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.sp805_wdog.clocktimer.timer**
A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.sp805_wdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.sp805_wdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.sp810_sysctrl**

Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.sp810_sysctrl.clkdiv_clk0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.sp810_sysctrl.clkdiv_clk1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.sp810_sysctrl.clkdiv_clk2**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.sp810_sysctrl.clkdiv_clk3**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.sram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.terminal_0**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.terminal_1**

Telnet terminal interface.
Type: `TelnetTerminal`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.terminal_2**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.terminal_3**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.trusted_key_storage**

Trusted Root-Key Storage unit.
Type: `RootKeyStorage`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.trusted_nv_counter**

Trusted Non-Volatile Counter unit.
Type: `NonVolatileCounter`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.trusted_rng**

Random Number Generator unit.
Type: `RandomNumberGenerator`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.trusted_watchdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.trusted_watchdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.trusted_watchdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.trusted_watchdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.trusted_watchdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.tzc_400**

TrustZone Address Space Controller.
Type: `TZC_400`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.ve_sysregs**

Type: `VE_SysRegs`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.vedcc**

Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.virtio_net**

VirtioNet device over MMIO transport.
Type: `VirtioNetMMIO`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.virtio_net_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.virtio_rng**

VirtioEntropy device over MMIO transport.
Type: `VirtioEntropyMMIO`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.virtioblockdevice**

VirtioBlock device over MMIO transport.
Type: `VirtioBlockMMIO`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.virtioblockdevice_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.virtiop9device**

virtio P9 server.
Type: `VirtioP9Device`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.virtiop9device_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.vis**

Display window for VE using Visualisation library.
Type: `VEVisualisation`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.bp.vram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.cci400**

Cache Coherent Interconnect for AXI4 ACE.
Type: `CCI400`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.clockdivider0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.cluster0**

ARM Cortex-A65AE_Cortex-A76AE Cluster CT model.
Type: `Cluster_ARM_Cortex-A65AE_Cortex-A76AE`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.cluster0.DSU.shared_cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.cluster0.subcluster0**

ARM Cortex-A65AE Cluster CT model.
Type: `Subcluster_ARM_Cortex-A65AE`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.cluster0.subcluster0.cpu0.thread0**

ARM Cortex-A65AE CT model.
Type: `ARM_Cortex-A65AE`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.cluster0.subcluster0.cpu0.thread1**

ARM Cortex-A65AE CT model.
Type: `ARM_Cortex-A65AE`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.cluster0.subcluster0.cpu1.thread0**

ARM Cortex-A65AE CT model.
Type: `ARM_Cortex-A65AE`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.cluster0.subcluster0.cpu1.thread1**

ARM Cortex-A65AE CT model.
Type: `ARM_Cortex-A65AE`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.cluster0.subcluster1**

ARM Cortex-A76AE Cluster CT model.
Type: `Subcluster_ARM_Cortex-A76AE`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.cluster0.subcluster1.cpu0**

ARM Cortex-A76AE CT model.
Type: `ARM_Cortex-A76AE`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.cluster0.subcluster1.cpu0.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.cluster0.subcluster1.cpu0.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.cluster0.subcluster1.cpu0.l2cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.cluster0.subcluster1.cpu1**

ARM Cortex-A76AE CT model.
Type: `ARM_Cortex-A76AE`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.cluster0.subcluster1.cpu1.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.cluster0.subcluster1.cpu1.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.cluster0.subcluster1.cpu1.l2cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.cluster0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.dapmemlogger**

Bus Logger.
Type: `PVBusLogger`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.elfloader**

ELF loader component.
Type: `ElfLoader`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.gic_distributor**

GIC Interrupt Redistribution Infrastucture component.
Type: `GIC_IRI`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.pctl**

Base Platforms Power Controller.
Type: `Base_PowerController`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.pctl.timer_reset**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.pctl.timer_reset.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.pctl.timer_reset.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `SchedulerThread`.

**`FVP_Base_Cortex_A65AEx2_Cortex_A76AEx2.pctl.timer_reset.timer.thread_event`**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

# 13.40 FVP_Base_Cortex-A65AEx4+Cortex-A76AEx4

List of instances in FVP_Base_Cortex-A65AEx4+Cortex-A76AEx4.

## FVP_Base_Cortex-A65AEx4+Cortex-A76AEx4 instances

**`FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4`**

Base Platform Compute Subsystem for ARMCortexA65AEx4CT_CortexA76AEx4CT.
Type: `FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4`.

**`FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp`**

Peripherals and address map for the Base Platform.
Type: `BasePlatformPeripherals`.

**`FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.Timer_0_1`**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**`FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.Timer_0_1.clk_div0`**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**`FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.Timer_0_1.clk_div1`**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**`FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.Timer_0_1.counter0`**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**`FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.Timer_0_1.counter1`**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**`FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.Timer_2_3`**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.Timer_2_3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.Timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.Timer_2_3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.Timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.ap_refclk**

ARM Generic Timer.
Type: `MemoryMappedGenericTimer`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.clock300MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.clock32KHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.dmc**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.dmc_phy**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.dummy_local_dap_rom**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.dummy_ram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.dummy_usb**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.exclusive_monitor**
Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.flash0**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.flash1**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.flashloader0**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.flashloader1**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.generic_watchdog**
ARM Generic Watchdog.
Type: `MemoryMappedGenericWatchdog`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.hdlcd0**
ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).
Type: `PL370_HDLCD`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.hdlcd0.timer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.hdlcd0.timer.timer**
A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.hdlcd0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.hdlcd0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.hdlcd0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.mmc**

Generic Multimedia Card.
Type: `MMC`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.mpe**

Encrypt memory transactions for each encryption context with an independent key to prevent mismatch access.
Type: `PVMemoryProtectionEngine`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.nontrustedrom**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.nontrustedromloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.ns_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.pl111_clcd_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.pl180_mci**

ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.ps2keyboard**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.ps2keyboard.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.ps2keyboard.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.ps2keyboard.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.ps2keyboard.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.ps2mouse.ps2_clocktimer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.ps2mouse.ps2_clocktimer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.psram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.refcounter**
Memory Mapped Counter Module for Generic Timers.
Type: `MemoryMappedCounterModule`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.reset_or**
Or Gate.
Type: `OrGate`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.rl_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.rt_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.s_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.secureDRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.secureSRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.secureflash**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.secureflashloader**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.smsc_91c111**

SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.sp805_wdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.sp805_wdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.sp805_wdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.sp805_wdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.sp805_wdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.sp810_sysctrl**

Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.sp810_sysctrl.clkdiv_clk0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.sp810_sysctrl.clkdiv_clk1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.sp810_sysctrl.clkdiv_clk2**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.sp810_sysctrl.clkdiv_clk3**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.sram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.terminal_0**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.terminal_1**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.terminal_2**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.terminal_3**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.trusted_key_storage**

Trusted Root-Key Storage unit.
Type: `RootKeyStorage`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.trusted_nv_counter**

Trusted Non-Volatile Counter unit.
Type: `NonVolatileCounter`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.trusted_rng**

Random Number Generator unit.
Type: `RandomNumberGenerator`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.trusted_watchdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.trusted_watchdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.trusted_watchdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.trusted_watchdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.trusted_watchdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.tzc_400**

TrustZone Address Space Controller.
Type: `TZC_400`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.ve_sysregs**

Type: `VE_SysRegs`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.vedcc**

Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.virtio_net**

VirtioNet device over MMIO transport.
Type: `VirtioNetMMIO`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.virtio_net_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.virtio_rng**

VirtioEntropy device over MMIO transport.
Type: `VirtioEntropyMMIO`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.virtioblockdevice**

VirtioBlock device over MMIO transport.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `VirtioBlockMMIO`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.virtioblockdevice_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.virtiop9device**

virtio P9 server.
Type: `VirtioP9Device`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.virtiop9device_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.vis**

Display window for VE using Visualisation library.
Type: `VEVisualisation`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.bp.vram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.cci400**

Cache Coherent Interconnect for AXI4 ACE.
Type: `CCI400`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.clockdivider0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.cluster0**

ARM Cortex-A65AE_Cortex-A76AE Cluster CT model.
Type: `Cluster_ARM_Cortex-A65AE_Cortex-A76AE`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.cluster0.DSU.shared_cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.cluster0.subcluster0**

ARM Cortex-A65AE Cluster CT model.
Type: `Subcluster_ARM_Cortex-A65AE`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.cluster0.subcluster0.cpu0.thread0**

ARM Cortex-A65AE CT model.
Type: `ARM_Cortex-A65AE`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.cluster0.subcluster0.cpu0.thread1**

ARM Cortex-A65AE CT model.
Type: `ARM_Cortex-A65AE`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.cluster0.subcluster0.cpu1.thread0**

ARM Cortex-A65AE CT model.
Type: `ARM_Cortex-A65AE`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.cluster0.subcluster0.cpu1.thread1**

ARM Cortex-A65AE CT model.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ARM_Cortex-A65AE`.

**`FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.cluster0.subcluster0.cpu2.thread0`**

ARM Cortex-A65AE CT model.
Type: `ARM_Cortex-A65AE`.

**`FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.cluster0.subcluster0.cpu2.thread1`**

ARM Cortex-A65AE CT model.
Type: `ARM_Cortex-A65AE`.

**`FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.cluster0.subcluster0.cpu3.thread0`**

ARM Cortex-A65AE CT model.
Type: `ARM_Cortex-A65AE`.

**`FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.cluster0.subcluster0.cpu3.thread1`**

ARM Cortex-A65AE CT model.
Type: `ARM_Cortex-A65AE`.

**`FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.cluster0.subcluster1`**

ARM Cortex-A76AE Cluster CT model.
Type: `Subcluster_ARM_Cortex-A76AE`.

**`FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.cluster0.subcluster1.cpu0`**

ARM Cortex-A76AE CT model.
Type: `ARM_Cortex-A76AE`.

**`FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.cluster0.subcluster1.cpu0.l1dcache`**

PV Cache.
Type: `PVCache`.

**`FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.cluster0.subcluster1.cpu0.l1icache`**

PV Cache.
Type: `PVCache`.

**`FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.cluster0.subcluster1.cpu0.l2cache`**

PV Cache.
Type: `PVCache`.

**`FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.cluster0.subcluster1.cpu1`**

ARM Cortex-A76AE CT model.
Type: `ARM_Cortex-A76AE`.

**`FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.cluster0.subcluster1.cpu1.l1dcache`**

PV Cache.
Type: `PVCache`.

**`FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.cluster0.subcluster1.cpu1.l1icache`**

PV Cache.
Type: `PVCache`.

**`FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.cluster0.subcluster1.cpu1.l2cache`**

PV Cache.
Type: `PVCache`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.cluster0.subcluster1.cpu2**

ARM Cortex-A76AE CT model.
Type: ARM_Cortex-A76AE.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.cluster0.subcluster1.cpu2.l1dcache**

PV Cache.
Type: PVCache.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.cluster0.subcluster1.cpu2.l1icache**

PV Cache.
Type: PVCache.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.cluster0.subcluster1.cpu2.l2cache**

PV Cache.
Type: PVCache.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.cluster0.subcluster1.cpu3**

ARM Cortex-A76AE CT model.
Type: ARM_Cortex-A76AE.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.cluster0.subcluster1.cpu3.l1dcache**

PV Cache.
Type: PVCache.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.cluster0.subcluster1.cpu3.l1icache**

PV Cache.
Type: PVCache.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.cluster0.subcluster1.cpu3.l2cache**

PV Cache.
Type: PVCache.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.cluster0_labeller**

Type: Labeller.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.dapmemlogger**

Bus Logger.
Type: PVBusLogger.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.elfloader**

ELF loader component.
Type: ElfLoader.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.gic_distributor**

GIC Interrupt Redistribution Infrastucture component.
Type: GIC_IRI.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.pctl**

Base Platforms Power Controller.
Type: Base_PowerController.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.pctl.timer_reset**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.pctl.timer_reset.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.pctl.timer_reset.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A65AEx4_Cortex_A76AEx4.pctl.timer_reset.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

## 13.41 FVP_Base_Cortex-A710

List of instances in FVP_Base_Cortex-A710.

### FVP_Base_Cortex-A710 instances

**FVP_Base_Cortex_A710**

Base Platform Compute Subsystem for ARMCortexA710CT.
Type: `FVP_Base_Cortex_A710`.

**FVP_Base_Cortex_A710.bp**

Peripherals and address map for the Base Platform.
Type: `BasePlatformPeripherals`.

**FVP_Base_Cortex_A710.bp.Timer_0_1**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Cortex_A710.bp.Timer_0_1.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A710.bp.Timer_0_1.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A710.bp.Timer_0_1.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A710.bp.Timer_0_1.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A710.bp.Timer_2_3**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Cortex_A710.bp.Timer_2_3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A710.bp.Timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A710.bp.Timer_2_3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A710.bp.Timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A710.bp.ap_refclk**

ARM Generic Timer.
Type: `MemoryMappedGenericTimer`.

**FVP_Base_Cortex_A710.bp.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A710.bp.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A710.bp.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A710.bp.clock300MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A710.bp.clock32KHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A710.bp.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A710.bp.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A710.bp.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A710.bp.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A710.bp.dmc**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A710.bp.dmc_phy**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A710.bp.dummy_local_dap_rom**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A710.bp.dummy_ram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A710.bp.dummy_usb**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A710.bp.exclusive_monitor**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_Base_Cortex_A710.bp.flash0**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A710.bp.flash1**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A710.bp.flashloader0**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A710.bp.flashloader1**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A710.bp.generic_watchdog**

ARM Generic Watchdog.
Type: `MemoryMappedGenericWatchdog`.

**FVP_Base_Cortex_A710.bp.hdlcd0**

ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `PL370_HDLCD`.

**FVP_Base_Cortex_A710.bp.hdlcd0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A710.bp.hdlcd0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A710.bp.hdlcd0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A710.bp.hdlcd0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A710.bp.hdlcd0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A710.bp.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_Base_Cortex_A710.bp.mmc**

Generic Multimedia Card.
Type: `MMC`.

**FVP_Base_Cortex_A710.bp.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A710.bp.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A710.bp.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A710.bp.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A710.bp.mpe**

Encrypt memory transactions for each encryption context with an independent key to prevent mismatch access.
Type: `PVMemoryProtectionEngine`.

**FVP_Base_Cortex_A710.bp.nontrustedrom**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A710.bp.nontrustedromloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A710.bp.ns_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A710.bp.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A710.bp.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A710.bp.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A710.bp.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A710.bp.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A710.bp.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A710.bp.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A710.bp.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A710.bp.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A710.bp.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A710.bp.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A710.bp.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A710.bp.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A710.bp.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A710.bp.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A710.bp.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A710.bp.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A710.bp.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A710.bp.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A710.bp.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A710.bp.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A710.bp.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A710.bp.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A710.bp.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A710.bp.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

**FVP_Base_Cortex_A710.bp.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A710.bp.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A710.bp.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A710.bp.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A710.bp.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_Base_Cortex_A710.bp.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A710.bp.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A710.bp.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A710.bp.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A710.bp.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A710.bp.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A710.bp.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A710.bp.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A710.bp.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_Base_Cortex_A710.bp.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

**FVP_Base_Cortex_A710.bp.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A710.bp.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A710.bp.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `SchedulerThread`.

**FVP_Base_Cortex_A710.bp.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A710.bp.pl111_clcd_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A710.bp.pl180_mci**

ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

**FVP_Base_Cortex_A710.bp.ps2keyboard**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

**FVP_Base_Cortex_A710.bp.ps2keyboard.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A710.bp.ps2keyboard.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A710.bp.ps2keyboard.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A710.bp.ps2keyboard.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A710.bp.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `PS2Mouse`.

**FVP_Base_Cortex_A710.bp.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A710.bp.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A710.bp.ps2mouse.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A710.bp.ps2mouse.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A710.bp.psram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A710.bp.refcounter**

Memory Mapped Counter Module for Generic Timers.
Type: `MemoryMappedCounterModule`.

**FVP_Base_Cortex_A710.bp.reset_or**

Or Gate.
Type: `OrGate`.

**FVP_Base_Cortex_A710.bp.rl_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A710.bp.rt_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A710.bp.s_dram**

RAM device, can be dynamic or static ram.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `RAMDevice`.

**FVP_Base_Cortex_A710.bp.secureDRAM**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A710.bp.secureSRAM**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A710.bp.secureflash**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A710.bp.secureflashloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A710.bp.smsc_91c111**

SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_Base_Cortex_A710.bp.sp805_wdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A710.bp.sp805_wdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A710.bp.sp805_wdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A710.bp.sp805_wdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A710.bp.sp805_wdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A710.bp.sp810_sysctrl**

Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_Base_Cortex_A710.bp.sp810_sysctrl.clkdiv_clk0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A710.bp.sp810_sysctrl.clkdiv_clk1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A710.bp.sp810_sysctrl.clkdiv_clk2**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A710.bp.sp810_sysctrl.clkdiv_clk3**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A710.bp.sram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A710.bp.terminal_0**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A710.bp.terminal_1**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A710.bp.terminal_2**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A710.bp.terminal_3**

Telnet terminal interface.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `TelnetTerminal`.

**FVP_Base_Cortex_A710.bp.trusted_key_storage**

Trusted Root-Key Storage unit.
Type: `RootKeyStorage`.

**FVP_Base_Cortex_A710.bp.trusted_nv_counter**

Trusted Non-Volatile Counter unit.
Type: `NonVolatileCounter`.

**FVP_Base_Cortex_A710.bp.trusted_rng**

Random Number Generator unit.
Type: `RandomNumberGenerator`.

**FVP_Base_Cortex_A710.bp.trusted_watchdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A710.bp.trusted_watchdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A710.bp.trusted_watchdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A710.bp.trusted_watchdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A710.bp.trusted_watchdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A710.bp.tzc_400**

TrustZone Address Space Controller.
Type: `TZC_400`.

**FVP_Base_Cortex_A710.bp.ve_sysregs**

Type: `VE_SysRegs`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A710.bp.vedcc**

Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

**FVP_Base_Cortex_A710.bp.virtio_net**

VirtioNet device over MMIO transport.
Type: `VirtioNetMMIO`.

**FVP_Base_Cortex_A710.bp.virtio_net_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A710.bp.virtio_rng**

VirtioEntropy device over MMIO transport.
Type: `VirtioEntropyMMIO`.

**FVP_Base_Cortex_A710.bp.virtioblockdevice**

VirtioBlock device over MMIO transport.
Type: `VirtioBlockMMIO`.

**FVP_Base_Cortex_A710.bp.virtioblockdevice_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A710.bp.virtiop9device**

virtio P9 server.
Type: `VirtioP9Device`.

**FVP_Base_Cortex_A710.bp.virtiop9device_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A710.bp.vis**

Display window for VE using Visualisation library.
Type: `VEVisualisation`.

**FVP_Base_Cortex_A710.bp.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_Base_Cortex_A710.bp.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A710.bp.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ClockTimerThread`.

**FVP_Base_Cortex_A710.bp.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A710.bp.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A710.bp.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A710.bp.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A710.bp.vram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A710.cci400**

Cache Coherent Interconnect for AXI4 ACE.
Type: `CCI400`.

**FVP_Base_Cortex_A710.clockdivider0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A710.cluster0**

ARM Cortex-A710 Cluster CT model.
Type: `Cluster_ARM_CortexA710`.

**FVP_Base_Cortex_A710.cluster0.DSU.shared_cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A710.cluster0.cpu0**

ARM Cortex-A710 CT model.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ARM_Cortex-A710`.

**`FVP_Base_Cortex_A710.cluster0.cpu0.dtlb`**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**`FVP_Base_Cortex_A710.cluster0.cpu0.l1dcache`**

PV Cache.
Type: `PVCache`.

**`FVP_Base_Cortex_A710.cluster0.cpu0.l1icache`**

PV Cache.
Type: `PVCache`.

**`FVP_Base_Cortex_A710.cluster0.cpu0.l2cache`**

PV Cache.
Type: `PVCache`.

**`FVP_Base_Cortex_A710.cluster0.cpu1`**

ARM Cortex-A710 CT model.
Type: `ARM_Cortex-A710`.

**`FVP_Base_Cortex_A710.cluster0.cpu1.dtlb`**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**`FVP_Base_Cortex_A710.cluster0.cpu1.l1dcache`**

PV Cache.
Type: `PVCache`.

**`FVP_Base_Cortex_A710.cluster0.cpu1.l1icache`**

PV Cache.
Type: `PVCache`.

**`FVP_Base_Cortex_A710.cluster0.cpu1.l2cache`**

PV Cache.
Type: `PVCache`.

**`FVP_Base_Cortex_A710.cluster0.cpu10`**

ARM Cortex-A710 CT model.
Type: `ARM_Cortex-A710`.

**`FVP_Base_Cortex_A710.cluster0.cpu10.dtlb`**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**`FVP_Base_Cortex_A710.cluster0.cpu10.l1dcache`**

PV Cache.
Type: `PVCache`.

**`FVP_Base_Cortex_A710.cluster0.cpu10.l1icache`**

PV Cache.
Type: `PVCache`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A710.cluster0.cpu10.l2cache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A710.cluster0.cpu11**
ARM Cortex-A710 CT model.
Type: `ARM_Cortex-A710`.

**FVP_Base_Cortex_A710.cluster0.cpu11.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A710.cluster0.cpu11.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A710.cluster0.cpu11.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A710.cluster0.cpu11.l2cache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A710.cluster0.cpu2**
ARM Cortex-A710 CT model.
Type: `ARM_Cortex-A710`.

**FVP_Base_Cortex_A710.cluster0.cpu2.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A710.cluster0.cpu2.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A710.cluster0.cpu2.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A710.cluster0.cpu2.l2cache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A710.cluster0.cpu3**
ARM Cortex-A710 CT model.
Type: `ARM_Cortex-A710`.

**FVP_Base_Cortex_A710.cluster0.cpu3.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A710.cluster0.cpu3.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A710.cluster0.cpu3.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A710.cluster0.cpu3.l2cache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A710.cluster0.cpu4**
ARM Cortex-A710 CT model.
Type: `ARM_Cortex-A710`.

**FVP_Base_Cortex_A710.cluster0.cpu4.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A710.cluster0.cpu4.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A710.cluster0.cpu4.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A710.cluster0.cpu4.l2cache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A710.cluster0.cpu5**
ARM Cortex-A710 CT model.
Type: `ARM_Cortex-A710`.

**FVP_Base_Cortex_A710.cluster0.cpu5.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A710.cluster0.cpu5.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A710.cluster0.cpu5.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A710.cluster0.cpu5.l2cache**
PV Cache.
Type: `PVCache`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A710.cluster0.cpu6**

ARM Cortex-A710 CT model.
Type: `ARM_Cortex-A710`.

**FVP_Base_Cortex_A710.cluster0.cpu6.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A710.cluster0.cpu6.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A710.cluster0.cpu6.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A710.cluster0.cpu6.l2cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A710.cluster0.cpu7**

ARM Cortex-A710 CT model.
Type: `ARM_Cortex-A710`.

**FVP_Base_Cortex_A710.cluster0.cpu7.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A710.cluster0.cpu7.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A710.cluster0.cpu7.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A710.cluster0.cpu7.l2cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A710.cluster0.cpu8**

ARM Cortex-A710 CT model.
Type: `ARM_Cortex-A710`.

**FVP_Base_Cortex_A710.cluster0.cpu8.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A710.cluster0.cpu8.l1dcache**

PV Cache.
Type: `PVCache`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A710.cluster0.cpu8.l1icache**
PV Cache.
Type: PVCache.

**FVP_Base_Cortex_A710.cluster0.cpu8.l2cache**
PV Cache.
Type: PVCache.

**FVP_Base_Cortex_A710.cluster0.cpu9**
ARM Cortex-A710 CT model.
Type: ARM_Cortex-A710.

**FVP_Base_Cortex_A710.cluster0.cpu9.dtlb**
TLB - instruction, data or unified.
Type: TlbCadi.

**FVP_Base_Cortex_A710.cluster0.cpu9.l1dcache**
PV Cache.
Type: PVCache.

**FVP_Base_Cortex_A710.cluster0.cpu9.l1icache**
PV Cache.
Type: PVCache.

**FVP_Base_Cortex_A710.cluster0.cpu9.l2cache**
PV Cache.
Type: PVCache.

**FVP_Base_Cortex_A710.cluster0_labeller**
Type: Labeller.

**FVP_Base_Cortex_A710.dapmemlogger**
Bus Logger.
Type: PVBusLogger.

**FVP_Base_Cortex_A710.elfloader**
ELF loader component.
Type: ElfLoader.

**FVP_Base_Cortex_A710.gic_distributor**
GIC Interrupt Redistribution Infrastucture component.
Type: GIC_IRI.

**FVP_Base_Cortex_A710.pctl**
Base Platforms Power Controller.
Type: Base_PowerController.

**FVP_Base_Cortex_A710.pctl.timer_reset**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**`FVP_Base_Cortex_A710.pctl.timer_reset.timer`**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**`FVP_Base_Cortex_A710.pctl.timer_reset.timer.thread`**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**`FVP_Base_Cortex_A710.pctl.timer_reset.timer.thread_event`**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

## 13.42 FVP_Base_Cortex-A715

List of instances in FVP_Base_Cortex-A715.

### FVP_Base_Cortex-A715 instances

**`FVP_Base_Cortex_A715`**

Base Platform Compute Subsystem for ARMCortexA715CT.
Type: `FVP_Base_Cortex_A715`.

**`FVP_Base_Cortex_A715.bp`**

Peripherals and address map for the Base Platform.
Type: `BasePlatformPeripherals`.

**`FVP_Base_Cortex_A715.bp.Timer_0_1`**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**`FVP_Base_Cortex_A715.bp.Timer_0_1.clk_div0`**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**`FVP_Base_Cortex_A715.bp.Timer_0_1.clk_div1`**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A715.bp.Timer_0_1.counter0**
Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A715.bp.Timer_0_1.counter1**
Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A715.bp.Timer_2_3**
ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Cortex_A715.bp.Timer_2_3.clk_div0**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A715.bp.Timer_2_3.clk_div1**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A715.bp.Timer_2_3.counter0**
Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A715.bp.Timer_2_3.counter1**
Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A715.bp.ap_refclk**
ARM Generic Timer.
Type: `MemoryMappedGenericTimer`.

**FVP_Base_Cortex_A715.bp.audioout**
SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**FVP_Base_Cortex_A715.bp.clock100Hz**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A715.bp.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A715.bp.clock300MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A715.bp.clock32KHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A715.bp.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A715.bp.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A715.bp.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A715.bp.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A715.bp.dmc**

RAM device, can be dynamic or static ram.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `RAMDevice`.

**FVP_Base_Cortex_A715.bp.dmc_phy**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A715.bp.dummy_local_dap_rom**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A715.bp.dummy_ram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A715.bp.dummy_usb**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A715.bp.exclusive_monitor**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_Base_Cortex_A715.bp.flash0**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A715.bp.flash1**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A715.bp.flashloader0**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A715.bp.flashloader1**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A715.bp.generic_watchdog**

ARM Generic Watchdog.
Type: `MemoryMappedGenericWatchdog`.

**FVP_Base_Cortex_A715.bp.hdlcd0**

ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).
Type: `PL370_HDLCD`.

**FVP_Base_Cortex_A715.bp.hdlcd0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A715.bp.hdlcd0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A715.bp.hdlcd0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A715.bp.hdlcd0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A715.bp.hdlcd0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A715.bp.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_Base_Cortex_A715.bp.mmc**

Generic Multimedia Card.
Type: `MMC`.

**FVP_Base_Cortex_A715.bp.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A715.bp.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A715.bp.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A715.bp.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A715.bp.mpe**

Encrypt memory transactions for each encryption context with an independent key to prevent mismatch access.
Type: `PVMemoryProtectionEngine`.

**FVP_Base_Cortex_A715.bp.nontrustedrom**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A715.bp.nontrustedromloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A715.bp.ns_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A715.bp.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A715.bp.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A715.bp.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A715.bp.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A715.bp.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A715.bp.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A715.bp.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A715.bp.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A715.bp.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A715.bp.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A715.bp.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A715.bp.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A715.bp.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A715.bp.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A715.bp.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A715.bp.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A715.bp.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A715.bp.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A715.bp.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A715.bp.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A715.bp.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A715.bp.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A715.bp.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A715.bp.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A715.bp.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

**FVP_Base_Cortex_A715.bp.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A715.bp.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A715.bp.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A715.bp.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A715.bp.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_Base_Cortex_A715.bp.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A715.bp.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A715.bp.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A715.bp.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A715.bp.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A715.bp.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A715.bp.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A715.bp.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A715.bp.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_Base_Cortex_A715.bp.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

**FVP_Base_Cortex_A715.bp.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A715.bp.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A715.bp.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A715.bp.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A715.bp.pl111_clcd_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A715.bp.pl180_mci**

ARM PrimeCell Multimedia Card Interface (PL180).

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `PL180_MCI`.

**FVP_Base_Cortex_A715.bp.ps2keyboard**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

**FVP_Base_Cortex_A715.bp.ps2keyboard.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A715.bp.ps2keyboard.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A715.bp.ps2keyboard.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A715.bp.ps2keyboard.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A715.bp.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

**FVP_Base_Cortex_A715.bp.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A715.bp.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A715.bp.ps2mouse.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A715.bp.ps2mouse.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A715.bp.psram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A715.bp.refcounter**

Memory Mapped Counter Module for Generic Timers.
Type: `MemoryMappedCounterModule`.

**FVP_Base_Cortex_A715.bp.reset_or**

Or Gate.
Type: `OrGate`.

**FVP_Base_Cortex_A715.bp.rl_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A715.bp.rt_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A715.bp.s_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A715.bp.secureDRAM**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A715.bp.secureSRAM**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A715.bp.secureflash**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A715.bp.secureflashloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A715.bp.smsc_91c111**

SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_Base_Cortex_A715.bp.sp805_wdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A715.bp.sp805_wdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A715.bp.sp805_wdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A715.bp.sp805_wdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A715.bp.sp805_wdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A715.bp.sp810_sysctrl**

Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_Base_Cortex_A715.bp.sp810_sysctrl.clkdiv_clk0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A715.bp.sp810_sysctrl.clkdiv_clk1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A715.bp.sp810_sysctrl.clkdiv_clk2**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A715.bp.sp810_sysctrl.clkdiv_clk3**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A715.bp.sram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A715.bp.terminal_0**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A715.bp.terminal_1**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A715.bp.terminal_2**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A715.bp.terminal_3**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A715.bp.trusted_key_storage**

Trusted Root-Key Storage unit.
Type: `RootKeyStorage`.

**FVP_Base_Cortex_A715.bp.trusted_nv_counter**

Trusted Non-Volatile Counter unit.
Type: `NonVolatileCounter`.

**FVP_Base_Cortex_A715.bp.trusted_rng**

Random Number Generator unit.
Type: `RandomNumberGenerator`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A715.bp.trusted_watchdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A715.bp.trusted_watchdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A715.bp.trusted_watchdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A715.bp.trusted_watchdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A715.bp.trusted_watchdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A715.bp.tzc_400**

TrustZone Address Space Controller.
Type: `TZC_400`.

**FVP_Base_Cortex_A715.bp.ve_sysregs**

Type: `VE_SysRegs`.

**FVP_Base_Cortex_A715.bp.vedcc**

Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

**FVP_Base_Cortex_A715.bp.virtio_net**

VirtioNet device over MMIO transport.
Type: `VirtioNetMMIO`.

**FVP_Base_Cortex_A715.bp.virtio_net_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A715.bp.virtio_rng**

VirtioEntropy device over MMIO transport.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `VirtioEntropyMMIO`.

**FVP_Base_Cortex_A715.bp.virtioblockdevice**

VirtioBlock device over MMIO transport.
Type: `VirtioBlockMMIO`.

**FVP_Base_Cortex_A715.bp.virtioblockdevice_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A715.bp.virtiop9device**

virtio P9 server.
Type: `VirtioP9Device`.

**FVP_Base_Cortex_A715.bp.virtiop9device_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A715.bp.vis**

Display window for VE using Visualisation library.
Type: `VEVisualisation`.

**FVP_Base_Cortex_A715.bp.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_Base_Cortex_A715.bp.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A715.bp.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A715.bp.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A715.bp.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A715.bp.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A715.bp.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A715.bp.vram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A715.cci400**

Cache Coherent Interconnect for AXI4 ACE.
Type: `CCI400`.

**FVP_Base_Cortex_A715.clockdivider0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A715.cluster0**

ARM CortexA715 Cluster CT model.
Type: `Cluster_ARM_CortexA715`.

**FVP_Base_Cortex_A715.cluster0.DSU.shared_cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A715.cluster0.cpu0**

ARM CortexA715 CT model.
Type: `ARM_Cortex-A715`.

**FVP_Base_Cortex_A715.cluster0.cpu0.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A715.cluster0.cpu0.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A715.cluster0.cpu0.l1icache**

PV Cache.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `PVCache`.

**FVP_Base_Cortex_A715.cluster0.cpu0.l2cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A715.cluster0.cpu1**

ARM CortexA715 CT model.
Type: `ARM_Cortex-A715`.

**FVP_Base_Cortex_A715.cluster0.cpu1.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A715.cluster0.cpu1.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A715.cluster0.cpu1.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A715.cluster0.cpu1.l2cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A715.cluster0.cpu10**

ARM CortexA715 CT model.
Type: `ARM_Cortex-A715`.

**FVP_Base_Cortex_A715.cluster0.cpu10.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A715.cluster0.cpu10.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A715.cluster0.cpu10.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A715.cluster0.cpu10.l2cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A715.cluster0.cpu11**

ARM CortexA715 CT model.
Type: `ARM_Cortex-A715`.

**FVP_Base_Cortex_A715.cluster0.cpu11.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A715.cluster0.cpu11.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A715.cluster0.cpu11.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A715.cluster0.cpu11.l2cache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A715.cluster0.cpu2**
ARM CortexA715 CT model.
Type: `ARM_Cortex-A715`.

**FVP_Base_Cortex_A715.cluster0.cpu2.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A715.cluster0.cpu2.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A715.cluster0.cpu2.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A715.cluster0.cpu2.l2cache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A715.cluster0.cpu3**
ARM CortexA715 CT model.
Type: `ARM_Cortex-A715`.

**FVP_Base_Cortex_A715.cluster0.cpu3.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A715.cluster0.cpu3.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A715.cluster0.cpu3.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A715.cluster0.cpu3.l2cache**
PV Cache.
Type: `PVCache`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A715.cluster0.cpu4**
ARM CortexA715 CT model.
Type: `ARM_Cortex-A715`.

**FVP_Base_Cortex_A715.cluster0.cpu4.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A715.cluster0.cpu4.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A715.cluster0.cpu4.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A715.cluster0.cpu4.l2cache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A715.cluster0.cpu5**
ARM CortexA715 CT model.
Type: `ARM_Cortex-A715`.

**FVP_Base_Cortex_A715.cluster0.cpu5.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A715.cluster0.cpu5.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A715.cluster0.cpu5.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A715.cluster0.cpu5.l2cache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A715.cluster0.cpu6**
ARM CortexA715 CT model.
Type: `ARM_Cortex-A715`.

**FVP_Base_Cortex_A715.cluster0.cpu6.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A715.cluster0.cpu6.l1dcache**
PV Cache.
Type: `PVCache`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A715.cluster0.cpu6.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A715.cluster0.cpu6.l2cache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A715.cluster0.cpu7**
ARM CortexA715 CT model.
Type: `ARM_Cortex-A715`.

**FVP_Base_Cortex_A715.cluster0.cpu7.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A715.cluster0.cpu7.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A715.cluster0.cpu7.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A715.cluster0.cpu7.l2cache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A715.cluster0.cpu8**
ARM CortexA715 CT model.
Type: `ARM_Cortex-A715`.

**FVP_Base_Cortex_A715.cluster0.cpu8.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A715.cluster0.cpu8.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A715.cluster0.cpu8.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A715.cluster0.cpu8.l2cache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A715.cluster0.cpu9**
ARM CortexA715 CT model.
Type: `ARM_Cortex-A715`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A715.cluster0.cpu9.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A715.cluster0.cpu9.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A715.cluster0.cpu9.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A715.cluster0.cpu9.l2cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A715.cluster0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A715.dapmemlogger**

Bus Logger.
Type: `PVBusLogger`.

**FVP_Base_Cortex_A715.elfloader**

ELF loader component.
Type: `ElfLoader`.

**FVP_Base_Cortex_A715.gic_distributor**

GIC Interrupt Redistribution Infrastucture component.
Type: `GIC_IRI`.

**FVP_Base_Cortex_A715.pctl**

Base Platforms Power Controller.
Type: `Base_PowerController`.

**FVP_Base_Cortex_A715.pctl.timer_reset**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A715.pctl.timer_reset.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ClockTimerThread64`.

**`FVP_Base_Cortex_A715.pctl.timer_reset.timer.thread`**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**`FVP_Base_Cortex_A715.pctl.timer_reset.timer.thread_event`**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

# 13.43 FVP_Base_Cortex-A720

List of instances in FVP_Base_Cortex-A720.

## FVP_Base_Cortex-A720 instances

**`FVP_Base_Cortex_A720`**

Base Platform Compute Subsystem for ARMCortexA720CT.
Type: `FVP_Base_Cortex_A720`.

**`FVP_Base_Cortex_A720.bp`**

Peripherals and address map for the Base Platform.
Type: `BasePlatformPeripherals`.

**`FVP_Base_Cortex_A720.bp.Timer_0_1`**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**`FVP_Base_Cortex_A720.bp.Timer_0_1.clk_div0`**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**`FVP_Base_Cortex_A720.bp.Timer_0_1.clk_div1`**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**`FVP_Base_Cortex_A720.bp.Timer_0_1.counter0`**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**`FVP_Base_Cortex_A720.bp.Timer_0_1.counter1`**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A720.bp.Timer_2_3**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Cortex_A720.bp.Timer_2_3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A720.bp.Timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A720.bp.Timer_2_3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A720.bp.Timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A720.bp.ap_refclk**

ARM Generic Timer.
Type: `MemoryMappedGenericTimer`.

**FVP_Base_Cortex_A720.bp.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**FVP_Base_Cortex_A720.bp.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A720.bp.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A720.bp.clock300MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A720.bp.clock32KHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A720.bp.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A720.bp.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A720.bp.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A720.bp.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A720.bp.dmc**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A720.bp.dmc_phy**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A720.bp.dummy_local_dap_rom**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A720.bp.dummy_ram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A720.bp.dummy_usb**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A720.bp.exclusive_monitor**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_Base_Cortex_A720.bp.flash0**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A720.bp.flash1**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A720.bp.flashloader0**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A720.bp.flashloader1**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A720.bp.generic_watchdog**

ARM Generic Watchdog.
Type: `MemoryMappedGenericWatchdog`.

**FVP_Base_Cortex_A720.bp.hdlcd0**

ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).
Type: `PL370_HDLCD`.

**FVP_Base_Cortex_A720.bp.hdlcd0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A720.bp.hdlcd0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A720.bp.hdlcd0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A720.bp.hdlcd0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A720.bp.hdlcd0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A720.bp.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_Base_Cortex_A720.bp.mmc**

Generic Multimedia Card.
Type: `MMC`.

**FVP_Base_Cortex_A720.bp.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A720.bp.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A720.bp.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A720.bp.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A720.bp.mpe**

Encrypt memory transactions for each encryption context with an independent key to prevent mismatch access.
Type: `PVMemoryProtectionEngine`.

**FVP_Base_Cortex_A720.bp.nontrustedrom**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A720.bp.nontrustedromloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A720.bp.ns_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A720.bp.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A720.bp.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A720.bp.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A720.bp.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A720.bp.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A720.bp.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A720.bp.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A720.bp.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A720.bp.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A720.bp.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A720.bp.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A720.bp.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A720.bp.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A720.bp.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A720.bp.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A720.bp.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A720.bp.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A720.bp.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A720.bp.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A720.bp.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A720.bp.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ClockTimerThread`.

**FVP_Base_Cortex_A720.bp.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A720.bp.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A720.bp.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A720.bp.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

**FVP_Base_Cortex_A720.bp.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A720.bp.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A720.bp.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A720.bp.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A720.bp.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_Base_Cortex_A720.bp.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A720.bp.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A720.bp.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A720.bp.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A720.bp.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A720.bp.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A720.bp.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A720.bp.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A720.bp.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_Base_Cortex_A720.bp.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

**FVP_Base_Cortex_A720.bp.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A720.bp.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A720.bp.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A720.bp.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A720.bp.pl111_clcd_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A720.bp.pl180_mci**

ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

**FVP_Base_Cortex_A720.bp.ps2keyboard**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A720.bp.ps2keyboard.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A720.bp.ps2keyboard.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A720.bp.ps2keyboard.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A720.bp.ps2keyboard.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A720.bp.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

**FVP_Base_Cortex_A720.bp.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A720.bp.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A720.bp.ps2mouse.ps2_clocktimer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A720.bp.ps2mouse.ps2_clocktimer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A720.bp.psram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A720.bp.refcounter**
Memory Mapped Counter Module for Generic Timers.
Type: `MemoryMappedCounterModule`.

**FVP_Base_Cortex_A720.bp.reset_or**
Or Gate.
Type: `OrGate`.

**FVP_Base_Cortex_A720.bp.rl_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A720.bp.rt_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A720.bp.s_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A720.bp.secureDRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A720.bp.secureSRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A720.bp.secureflash**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A720.bp.secureflashloader**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A720.bp.smsc_91c111**

SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_Base_Cortex_A720.bp.sp805_wdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A720.bp.sp805_wdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A720.bp.sp805_wdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A720.bp.sp805_wdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A720.bp.sp805_wdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A720.bp.sp810_sysctrl**

Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_Base_Cortex_A720.bp.sp810_sysctrl.clkdiv_clk0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A720.bp.sp810_sysctrl.clkdiv_clk1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A720.bp.sp810_sysctrl.clkdiv_clk2**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A720.bp.sp810_sysctrl.clkdiv_clk3**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A720.bp.sram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A720.bp.terminal_0**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A720.bp.terminal_1**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A720.bp.terminal_2**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A720.bp.terminal_3**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A720.bp.trusted_key_storage**

Trusted Root-Key Storage unit.
Type: `RootKeyStorage`.

**FVP_Base_Cortex_A720.bp.trusted_nv_counter**

Trusted Non-Volatile Counter unit.
Type: `NonVolatileCounter`.

**FVP_Base_Cortex_A720.bp.trusted_rng**

Random Number Generator unit.
Type: `RandomNumberGenerator`.

**FVP_Base_Cortex_A720.bp.trusted_watchdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A720.bp.trusted_watchdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A720.bp.trusted_watchdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A720.bp.trusted_watchdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A720.bp.trusted_watchdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A720.bp.tzc_400**

TrustZone Address Space Controller.
Type: `TZC_400`.

**FVP_Base_Cortex_A720.bp.ve_sysregs**

Type: `VE_SysRegs`.

**FVP_Base_Cortex_A720.bp.vedcc**

Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

**FVP_Base_Cortex_A720.bp.virtio_net**

VirtioNet device over MMIO transport.
Type: `VirtioNetMMIO`.

**FVP_Base_Cortex_A720.bp.virtio_net_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A720.bp.virtio_rng**

VirtioEntropy device over MMIO transport.
Type: `VirtioEntropyMMIO`.

**FVP_Base_Cortex_A720.bp.virtioblockdevice**

VirtioBlock device over MMIO transport.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `VirtioBlockMMIO`.

**FVP_Base_Cortex_A720.bp.virtioblockdevice_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A720.bp.virtiop9device**

virtio P9 server.
Type: `VirtioP9Device`.

**FVP_Base_Cortex_A720.bp.virtiop9device_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A720.bp.vis**

Display window for VE using Visualisation library.
Type: `VEVisualisation`.

**FVP_Base_Cortex_A720.bp.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_Base_Cortex_A720.bp.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A720.bp.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A720.bp.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A720.bp.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A720.bp.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A720.bp.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A720.bp.vram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A720.cci400**

Cache Coherent Interconnect for AXI4 ACE.
Type: `CCI400`.

**FVP_Base_Cortex_A720.clockdivider0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A720.cluster0**

ARM Cortex-A720 Cluster CT model.
Type: `Cluster_ARM_Cortex-A720`.

**FVP_Base_Cortex_A720.cluster0.DSU.shared_cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A720.cluster0.cpu0**

ARM Cortex-A720 CT model.
Type: `ARM_Cortex-A720`.

**FVP_Base_Cortex_A720.cluster0.cpu0.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A720.cluster0.cpu0.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A720.cluster0.cpu0.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A720.cluster0.cpu0.l2cache**

PV Cache.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `PVCache`.

**FVP_Base_Cortex_A720.cluster0.cpu1**

ARM Cortex-A720 CT model.
Type: `ARM_Cortex-A720`.

**FVP_Base_Cortex_A720.cluster0.cpu1.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A720.cluster0.cpu1.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A720.cluster0.cpu1.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A720.cluster0.cpu1.l2cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A720.cluster0.cpu10**

ARM Cortex-A720 CT model.
Type: `ARM_Cortex-A720`.

**FVP_Base_Cortex_A720.cluster0.cpu10.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A720.cluster0.cpu10.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A720.cluster0.cpu10.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A720.cluster0.cpu10.l2cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A720.cluster0.cpu11**

ARM Cortex-A720 CT model.
Type: `ARM_Cortex-A720`.

**FVP_Base_Cortex_A720.cluster0.cpu11.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A720.cluster0.cpu11.l1dcache**

PV Cache.
Type: `PVCache`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A720.cluster0.cpu11.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A720.cluster0.cpu11.l2cache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A720.cluster0.cpu12**
ARM Cortex-A720 CT model.
Type: `ARM_Cortex-A720`.

**FVP_Base_Cortex_A720.cluster0.cpu12.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A720.cluster0.cpu12.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A720.cluster0.cpu12.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A720.cluster0.cpu12.l2cache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A720.cluster0.cpu13**
ARM Cortex-A720 CT model.
Type: `ARM_Cortex-A720`.

**FVP_Base_Cortex_A720.cluster0.cpu13.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A720.cluster0.cpu13.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A720.cluster0.cpu13.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A720.cluster0.cpu13.l2cache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A720.cluster0.cpu2**
ARM Cortex-A720 CT model.
Type: `ARM_Cortex-A720`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A720.cluster0.cpu2.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A720.cluster0.cpu2.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A720.cluster0.cpu2.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A720.cluster0.cpu2.l2cache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A720.cluster0.cpu3**
ARM Cortex-A720 CT model.
Type: `ARM_Cortex-A720`.

**FVP_Base_Cortex_A720.cluster0.cpu3.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A720.cluster0.cpu3.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A720.cluster0.cpu3.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A720.cluster0.cpu3.l2cache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A720.cluster0.cpu4**
ARM Cortex-A720 CT model.
Type: `ARM_Cortex-A720`.

**FVP_Base_Cortex_A720.cluster0.cpu4.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A720.cluster0.cpu4.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A720.cluster0.cpu4.l1icache**
PV Cache.
Type: `PVCache`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A720.cluster0.cpu4.l2cache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A720.cluster0.cpu5**
ARM Cortex-A720 CT model.
Type: `ARM_Cortex-A720`.

**FVP_Base_Cortex_A720.cluster0.cpu5.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A720.cluster0.cpu5.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A720.cluster0.cpu5.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A720.cluster0.cpu5.l2cache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A720.cluster0.cpu6**
ARM Cortex-A720 CT model.
Type: `ARM_Cortex-A720`.

**FVP_Base_Cortex_A720.cluster0.cpu6.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A720.cluster0.cpu6.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A720.cluster0.cpu6.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A720.cluster0.cpu6.l2cache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A720.cluster0.cpu7**
ARM Cortex-A720 CT model.
Type: `ARM_Cortex-A720`.

**FVP_Base_Cortex_A720.cluster0.cpu7.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A720.cluster0.cpu7.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A720.cluster0.cpu7.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A720.cluster0.cpu7.l2cache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A720.cluster0.cpu8**
ARM Cortex-A720 CT model.
Type: `ARM_Cortex-A720`.

**FVP_Base_Cortex_A720.cluster0.cpu8.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A720.cluster0.cpu8.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A720.cluster0.cpu8.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A720.cluster0.cpu8.l2cache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A720.cluster0.cpu9**
ARM Cortex-A720 CT model.
Type: `ARM_Cortex-A720`.

**FVP_Base_Cortex_A720.cluster0.cpu9.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A720.cluster0.cpu9.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A720.cluster0.cpu9.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A720.cluster0.cpu9.l2cache**
PV Cache.
Type: `PVCache`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A720.cluster0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A720.dapmemlogger**

Bus Logger.
Type: `PVBusLogger`.

**FVP_Base_Cortex_A720.elfloader**

ELF loader component.
Type: `ElfLoader`.

**FVP_Base_Cortex_A720.gic_distributor**

GIC Interrupt Redistribution Infrastucture component.
Type: `GIC_IRI`.

**FVP_Base_Cortex_A720.pctl**

Base Platforms Power Controller.
Type: `Base_PowerController`.

**FVP_Base_Cortex_A720.pctl.timer_reset**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A720.pctl.timer_reset.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A720.pctl.timer_reset.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A720.pctl.timer_reset.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

# 13.44 FVP_Base_Cortex-A72x1

List of instances in FVP_Base_Cortex-A72x1.

## FVP_Base_Cortex-A72x1 instances

**FVP_Base_Cortex_A72x1**

Base Platform Compute Subsystem for ARMCortexA72x1CT.
Type: `FVP_Base_Cortex_A72x1`.

**FVP_Base_Cortex_A72x1.bp**

Peripherals and address map for the Base Platform.
Type: `BasePlatformPeripherals`.

**FVP_Base_Cortex_A72x1.bp.Timer_0_1**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Cortex_A72x1.bp.Timer_0_1.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x1.bp.Timer_0_1.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x1.bp.Timer_0_1.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A72x1.bp.Timer_0_1.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A72x1.bp.Timer_2_3**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Cortex_A72x1.bp.Timer_2_3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A72x1.bp.Timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x1.bp.Timer_2_3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A72x1.bp.Timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A72x1.bp.ap_refclk**

ARM Generic Timer.
Type: `MemoryMappedGenericTimer`.

**FVP_Base_Cortex_A72x1.bp.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**FVP_Base_Cortex_A72x1.bp.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x1.bp.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x1.bp.clock300MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x1.bp.clock32KHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A72x1.bp.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x1.bp.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x1.bp.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x1.bp.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x1.bp.dmc**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A72x1.bp.dmc_phy**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A72x1.bp.dummy_local_dap_rom**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A72x1.bp.dummy_ram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A72x1.bp.dummy_usb**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A72x1.bp.exclusive_monitor**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A72x1.bp.flash0**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A72x1.bp.flash1**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A72x1.bp.flashloader0**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A72x1.bp.flashloader1**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A72x1.bp.generic_watchdog**

ARM Generic Watchdog.
Type: `MemoryMappedGenericWatchdog`.

**FVP_Base_Cortex_A72x1.bp.hdlcd0**

ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).
Type: `PL370_HDLCD`.

**FVP_Base_Cortex_A72x1.bp.hdlcd0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A72x1.bp.hdlcd0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A72x1.bp.hdlcd0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A72x1.bp.hdlcd0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A72x1.bp.hdlcd0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A72x1.bp.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_Base_Cortex_A72x1.bp.mmc**

Generic Multimedia Card.
Type: `MMC`.

**FVP_Base_Cortex_A72x1.bp.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A72x1.bp.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A72x1.bp.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A72x1.bp.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A72x1.bp.mpe**

Encrypt memory transactions for each encryption context with an independent key to prevent mismatch access.
Type: `PVMemoryProtectionEngine`.

**FVP_Base_Cortex_A72x1.bp.nontrustedrom**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A72x1.bp.nontrustedromloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A72x1.bp.ns_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A72x1.bp.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A72x1.bp.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x1.bp.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A72x1.bp.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A72x1.bp.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A72x1.bp.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A72x1.bp.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A72x1.bp.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x1.bp.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A72x1.bp.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A72x1.bp.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A72x1.bp.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A72x1.bp.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A72x1.bp.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x1.bp.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ClockTimerThread`.

**FVP_Base_Cortex_A72x1.bp.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A72x1.bp.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A72x1.bp.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A72x1.bp.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A72x1.bp.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x1.bp.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A72x1.bp.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A72x1.bp.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A72x1.bp.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A72x1.bp.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

**FVP_Base_Cortex_A72x1.bp.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A72x1.bp.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A72x1.bp.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A72x1.bp.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A72x1.bp.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_Base_Cortex_A72x1.bp.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A72x1.bp.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A72x1.bp.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A72x1.bp.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A72x1.bp.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A72x1.bp.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x1.bp.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A72x1.bp.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x1.bp.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_Base_Cortex_A72x1.bp.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A72x1.bp.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A72x1.bp.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A72x1.bp.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A72x1.bp.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A72x1.bp.pl111_clcd_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A72x1.bp.pl180_mci**

ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

**FVP_Base_Cortex_A72x1.bp.ps2keyboard**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

**FVP_Base_Cortex_A72x1.bp.ps2keyboard.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A72x1.bp.ps2keyboard.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A72x1.bp.ps2keyboard.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A72x1.bp.ps2keyboard.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A72x1.bp.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

**FVP_Base_Cortex_A72x1.bp.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A72x1.bp.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A72x1.bp.ps2mouse.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A72x1.bp.ps2mouse.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A72x1.bp.psram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A72x1.bp.refcounter**
Memory Mapped Counter Module for Generic Timers.
Type: `MemoryMappedCounterModule`.

**FVP_Base_Cortex_A72x1.bp.reset_or**
Or Gate.
Type: `OrGate`.

**FVP_Base_Cortex_A72x1.bp.rl_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A72x1.bp.rt_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A72x1.bp.s_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A72x1.bp.secureDRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A72x1.bp.secureSRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A72x1.bp.secureflash**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A72x1.bp.secureflashloader**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A72x1.bp.smsc_91c111**
SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_Base_Cortex_A72x1.bp.sp805_wdog**
ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A72x1.bp.sp805_wdog.clocktimer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A72x1.bp.sp805_wdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A72x1.bp.sp805_wdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A72x1.bp.sp805_wdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A72x1.bp.sp810_sysctrl**

Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_Base_Cortex_A72x1.bp.sp810_sysctrl.clkdiv_clk0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x1.bp.sp810_sysctrl.clkdiv_clk1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x1.bp.sp810_sysctrl.clkdiv_clk2**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x1.bp.sp810_sysctrl.clkdiv_clk3**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**`FVP_Base_Cortex_A72x1.bp.sram`**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**`FVP_Base_Cortex_A72x1.bp.terminal_0`**
Telnet terminal interface.
Type: `TelnetTerminal`.

**`FVP_Base_Cortex_A72x1.bp.terminal_1`**
Telnet terminal interface.
Type: `TelnetTerminal`.

**`FVP_Base_Cortex_A72x1.bp.terminal_2`**
Telnet terminal interface.
Type: `TelnetTerminal`.

**`FVP_Base_Cortex_A72x1.bp.terminal_3`**
Telnet terminal interface.
Type: `TelnetTerminal`.

**`FVP_Base_Cortex_A72x1.bp.trusted_key_storage`**
Trusted Root-Key Storage unit.
Type: `RootKeyStorage`.

**`FVP_Base_Cortex_A72x1.bp.trusted_nv_counter`**
Trusted Non-Volatile Counter unit.
Type: `NonVolatileCounter`.

**`FVP_Base_Cortex_A72x1.bp.trusted_rng`**
Random Number Generator unit.
Type: `RandomNumberGenerator`.

**`FVP_Base_Cortex_A72x1.bp.trusted_watchdog`**
ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**`FVP_Base_Cortex_A72x1.bp.trusted_watchdog.clocktimer`**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**`FVP_Base_Cortex_A72x1.bp.trusted_watchdog.clocktimer.timer`**
A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A72x1.bp.trusted_watchdog.clocktimer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A72x1.bp.trusted_watchdog.clocktimer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A72x1.bp.tzc_400**
TrustZone Address Space Controller.
Type: `TZC_400`.

**FVP_Base_Cortex_A72x1.bp.ve_sysregs**
Type: `VE_SysRegs`.

**FVP_Base_Cortex_A72x1.bp.vedcc**
Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

**FVP_Base_Cortex_A72x1.bp.virtio_net**
VirtioNet device over MMIO transport.
Type: `VirtioNetMMIO`.

**FVP_Base_Cortex_A72x1.bp.virtio_net_labeller**
Type: `Labeller`.

**FVP_Base_Cortex_A72x1.bp.virtio_rng**
VirtioEntropy device over MMIO transport.
Type: `VirtioEntropyMMIO`.

**FVP_Base_Cortex_A72x1.bp.virtioblockdevice**
VirtioBlock device over MMIO transport.
Type: `VirtioBlockMMIO`.

**FVP_Base_Cortex_A72x1.bp.virtioblockdevice_labeller**
Type: `Labeller`.

**FVP_Base_Cortex_A72x1.bp.virtiop9device**
virtio P9 server.
Type: `VirtioP9Device`.

**FVP_Base_Cortex_A72x1.bp.virtiop9device_labeller**
Type: `Labeller`.

**FVP_Base_Cortex_A72x1.bp.vis**
Display window for VE using Visualisation library.
Type: `VEVisualisation`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A72x1.bp.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_Base_Cortex_A72x1.bp.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x1.bp.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A72x1.bp.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A72x1.bp.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A72x1.bp.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A72x1.bp.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x1.bp.vram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A72x1.cci400**

Cache Coherent Interconnect for AXI4 ACE.
Type: `CCI400`.

**FVP_Base_Cortex_A72x1.clockdivider0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x1.cluster0**

ARM Cortex-A72 Cluster CT model.
Type: `Cluster_ARM_Cortex-A72`.

**FVP_Base_Cortex_A72x1.cluster0.cpu0**

ARM Cortex-A72 CT model.
Type: `ARM_Cortex-A72`.

**FVP_Base_Cortex_A72x1.cluster0.cpu0.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A72x1.cluster0.cpu0.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A72x1.cluster0.cpu0.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A72x1.cluster0.l2_cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A72x1.cluster0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A72x1.dapmemlogger**

Bus Logger.
Type: `PVBusLogger`.

**FVP_Base_Cortex_A72x1.elfloader**

ELF loader component.
Type: `ElfLoader`.

**FVP_Base_Cortex_A72x1.gic_distributor**

GIC Interrupt Redistribution Infrastucture component.
Type: `GIC_IRI`.

**FVP_Base_Cortex_A72x1.pctl**

Base Platforms Power Controller.
Type: `Base_PowerController`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A72x1.pctl.timer_reset**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A72x1.pctl.timer_reset.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A72x1.pctl.timer_reset.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A72x1.pctl.timer_reset.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

## 13.45 FVP_Base_Cortex-A72x1-A53x1

List of instances in FVP_Base_Cortex-A72x1-A53x1.

### FVP_Base_Cortex-A72x1-A53x1 instances

**FVP_Base_Cortex_A72x1_A53x1**

Base Platform Compute Subsystem for ARMCortexA72x1CT and ARMCortexA53x1CT.
Type: `FVP_Base_Cortex_A72x1_A53x1`.

**FVP_Base_Cortex_A72x1_A53x1.bp**

Peripherals and address map for the Base Platform.
Type: `BasePlatformPeripherals`.

**FVP_Base_Cortex_A72x1_A53x1.bp.Timer_0_1**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Cortex_A72x1_A53x1.bp.Timer_0_1.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x1_A53x1.bp.Timer_0_1.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x1_A53x1.bp.Timer_0_1.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A72x1_A53x1.bp.Timer_0_1.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A72x1_A53x1.bp.Timer_2_3**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Cortex_A72x1_A53x1.bp.Timer_2_3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x1_A53x1.bp.Timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x1_A53x1.bp.Timer_2_3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A72x1_A53x1.bp.Timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A72x1_A53x1.bp.ap_refclk**

ARM Generic Timer.
Type: `MemoryMappedGenericTimer`.

**FVP_Base_Cortex_A72x1_A53x1.bp.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A72x1_A53x1.bp.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x1_A53x1.bp.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x1_A53x1.bp.clock300MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x1_A53x1.bp.clock32KHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x1_A53x1.bp.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x1_A53x1.bp.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x1_A53x1.bp.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A72x1_A53x1.bp.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x1_A53x1.bp.dmc**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A72x1_A53x1.bp.dmc_phy**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A72x1_A53x1.bp.dummy_local_dap_rom**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A72x1_A53x1.bp.dummy_ram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A72x1_A53x1.bp.dummy_usb**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A72x1_A53x1.bp.exclusive_monitor**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_Base_Cortex_A72x1_A53x1.bp.flash0**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A72x1_A53x1.bp.flash1**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A72x1_A53x1.bp.flashloader0**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A72x1_A53x1.bp.flashloader1**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A72x1_A53x1.bp.generic_watchdog**

ARM Generic Watchdog.
Type: `MemoryMappedGenericWatchdog`.

**FVP_Base_Cortex_A72x1_A53x1.bp.hdlcd0**

ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `PL370_HDLCD`.

**FVP_Base_Cortex_A72x1_A53x1.bp.hdlcd0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A72x1_A53x1.bp.hdlcd0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A72x1_A53x1.bp.hdlcd0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A72x1_A53x1.bp.hdlcd0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A72x1_A53x1.bp.hdlcd0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A72x1_A53x1.bp.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_Base_Cortex_A72x1_A53x1.bp.mmc**

Generic Multimedia Card.
Type: `MMC`.

**FVP_Base_Cortex_A72x1_A53x1.bp.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A72x1_A53x1.bp.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A72x1_A53x1.bp.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A72x1_A53x1.bp.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A72x1_A53x1.bp.mpe**

Encrypt memory transactions for each encryption context with an independent key to prevent mismatch access.
Type: `PVMemoryProtectionEngine`.

**FVP_Base_Cortex_A72x1_A53x1.bp.nontrustedrom**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A72x1_A53x1.bp.nontrustedromloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A72x1_A53x1.bp.ns_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A72x1_A53x1.bp.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A72x1_A53x1.bp.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x1_A53x1.bp.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A72x1_A53x1.bp.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A72x1_A53x1.bp.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A72x1_A53x1.bp.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A72x1_A53x1.bp.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A72x1_A53x1.bp.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x1_A53x1.bp.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A72x1_A53x1.bp.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A72x1_A53x1.bp.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A72x1_A53x1.bp.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A72x1_A53x1.bp.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A72x1_A53x1.bp.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x1_A53x1.bp.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A72x1_A53x1.bp.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A72x1_A53x1.bp.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A72x1_A53x1.bp.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A72x1_A53x1.bp.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A72x1_A53x1.bp.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x1_A53x1.bp.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A72x1_A53x1.bp.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A72x1_A53x1.bp.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A72x1_A53x1.bp.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A72x1_A53x1.bp.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

**FVP_Base_Cortex_A72x1_A53x1.bp.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A72x1_A53x1.bp.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A72x1_A53x1.bp.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A72x1_A53x1.bp.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A72x1_A53x1.bp.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_Base_Cortex_A72x1_A53x1.bp.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A72x1_A53x1.bp.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A72x1_A53x1.bp.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A72x1_A53x1.bp.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A72x1_A53x1.bp.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A72x1_A53x1.bp.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x1_A53x1.bp.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A72x1_A53x1.bp.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x1_A53x1.bp.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_Base_Cortex_A72x1_A53x1.bp.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

**FVP_Base_Cortex_A72x1_A53x1.bp.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A72x1_A53x1.bp.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A72x1_A53x1.bp.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `SchedulerThread`.

**FVP_Base_Cortex_A72x1_A53x1.bp.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A72x1_A53x1.bp.pl111_clcd_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A72x1_A53x1.bp.pl180_mci**

ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

**FVP_Base_Cortex_A72x1_A53x1.bp.ps2keyboard**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

**FVP_Base_Cortex_A72x1_A53x1.bp.ps2keyboard.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A72x1_A53x1.bp.ps2keyboard.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A72x1_A53x1.bp.ps2keyboard.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A72x1_A53x1.bp.ps2keyboard.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A72x1_A53x1.bp.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `PS2Mouse`.

**FVP_Base_Cortex_A72x1_A53x1.bp.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A72x1_A53x1.bp.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A72x1_A53x1.bp.ps2mouse.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A72x1_A53x1.bp.ps2mouse.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A72x1_A53x1.bp.psram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A72x1_A53x1.bp.refcounter**

Memory Mapped Counter Module for Generic Timers.
Type: `MemoryMappedCounterModule`.

**FVP_Base_Cortex_A72x1_A53x1.bp.reset_or**

Or Gate.
Type: `OrGate`.

**FVP_Base_Cortex_A72x1_A53x1.bp.rl_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A72x1_A53x1.bp.rt_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A72x1_A53x1.bp.s_dram**

RAM device, can be dynamic or static ram.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `RAMDevice`.

**FVP_Base_Cortex_A72x1_A53x1.bp.secureDRAM**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A72x1_A53x1.bp.secureSRAM**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A72x1_A53x1.bp.secureflash**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A72x1_A53x1.bp.secureflashloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A72x1_A53x1.bp.smsc_91c111**

SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_Base_Cortex_A72x1_A53x1.bp.sp805_wdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A72x1_A53x1.bp.sp805_wdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A72x1_A53x1.bp.sp805_wdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A72x1_A53x1.bp.sp805_wdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A72x1_A53x1.bp.sp805_wdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A72x1_A53x1.bp.sp810_sysctrl**

Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_Base_Cortex_A72x1_A53x1.bp.sp810_sysctrl.clkdiv_clk0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x1_A53x1.bp.sp810_sysctrl.clkdiv_clk1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x1_A53x1.bp.sp810_sysctrl.clkdiv_clk2**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x1_A53x1.bp.sp810_sysctrl.clkdiv_clk3**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x1_A53x1.bp.sram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A72x1_A53x1.bp.terminal_0**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A72x1_A53x1.bp.terminal_1**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A72x1_A53x1.bp.terminal_2**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A72x1_A53x1.bp.terminal_3**

Telnet terminal interface.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `TelnetTerminal`.

**FVP_Base_Cortex_A72x1_A53x1.bp.trusted_key_storage**

Trusted Root-Key Storage unit.
Type: `RootKeyStorage`.

**FVP_Base_Cortex_A72x1_A53x1.bp.trusted_nv_counter**

Trusted Non-Volatile Counter unit.
Type: `NonVolatileCounter`.

**FVP_Base_Cortex_A72x1_A53x1.bp.trusted_rng**

Random Number Generator unit.
Type: `RandomNumberGenerator`.

**FVP_Base_Cortex_A72x1_A53x1.bp.trusted_watchdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A72x1_A53x1.bp.trusted_watchdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A72x1_A53x1.bp.trusted_watchdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A72x1_A53x1.bp.trusted_watchdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A72x1_A53x1.bp.trusted_watchdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A72x1_A53x1.bp.tzc_400**

TrustZone Address Space Controller.
Type: `TZC_400`.

**FVP_Base_Cortex_A72x1_A53x1.bp.ve_sysregs**

Type: `VE_SysRegs`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A72x1_A53x1.bp.vedcc**

Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

**FVP_Base_Cortex_A72x1_A53x1.bp.virtio_net**

VirtioNet device over MMIO transport.
Type: `VirtioNetMMIO`.

**FVP_Base_Cortex_A72x1_A53x1.bp.virtio_net_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A72x1_A53x1.bp.virtio_rng**

VirtioEntropy device over MMIO transport.
Type: `VirtioEntropyMMIO`.

**FVP_Base_Cortex_A72x1_A53x1.bp.virtioblockdevice**

VirtioBlock device over MMIO transport.
Type: `VirtioBlockMMIO`.

**FVP_Base_Cortex_A72x1_A53x1.bp.virtioblockdevice_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A72x1_A53x1.bp.virtiop9device**

virtio P9 server.
Type: `VirtioP9Device`.

**FVP_Base_Cortex_A72x1_A53x1.bp.virtiop9device_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A72x1_A53x1.bp.vis**

Display window for VE using Visualisation library.
Type: `VEVisualisation`.

**FVP_Base_Cortex_A72x1_A53x1.bp.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_Base_Cortex_A72x1_A53x1.bp.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x1_A53x1.bp.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ClockTimerThread`.

**FVP_Base_Cortex_A72x1_A53x1.bp.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A72x1_A53x1.bp.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A72x1_A53x1.bp.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A72x1_A53x1.bp.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x1_A53x1.bp.vram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A72x1_A53x1.cci400**

Cache Coherent Interconnect for AXI4 ACE.
Type: `CCI400`.

**FVP_Base_Cortex_A72x1_A53x1.clockdivider0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x1_A53x1.clockdivider1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x1_A53x1.cluster0**

ARM Cortex-A72 Cluster CT model.
Type: `Cluster_ARM_Cortex-A72`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A72x1_A53x1.cluster0.cpu0**
ARM Cortex-A72 CT model.
Type: `ARM_Cortex-A72`.

**FVP_Base_Cortex_A72x1_A53x1.cluster0.cpu0.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A72x1_A53x1.cluster0.cpu0.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A72x1_A53x1.cluster0.cpu0.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A72x1_A53x1.cluster0.l2_cache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A72x1_A53x1.cluster0_labeller**
Type: `Labeller`.

**FVP_Base_Cortex_A72x1_A53x1.cluster1**
ARM Cortex-A53 Cluster CT model.
Type: `Cluster_ARM_Cortex-A53`.

**FVP_Base_Cortex_A72x1_A53x1.cluster1.cpu0**
ARM Cortex-A53 CT model.
Type: `ARM_Cortex-A53`.

**FVP_Base_Cortex_A72x1_A53x1.cluster1.cpu0.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A72x1_A53x1.cluster1.cpu0.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A72x1_A53x1.cluster1.cpu0.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A72x1_A53x1.cluster1.l2_cache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A72x1_A53x1.cluster1_labeller**
Type: `Labeller`.

**FVP_Base_Cortex_A72x1_A53x1.dapmemlogger**
Bus Logger.
Type: `PVBusLogger`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A72x1_A53x1.elfloader**

ELF loader component.
Type: `ElfLoader`.

**FVP_Base_Cortex_A72x1_A53x1.gic_distributor**

GIC Interrupt Redistribution Infrastucture component.
Type: `GIC_IRI`.

**FVP_Base_Cortex_A72x1_A53x1.pctl**

Base Platforms Power Controller.
Type: `Base_PowerController`.

**FVP_Base_Cortex_A72x1_A53x1.pctl.timer_reset**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A72x1_A53x1.pctl.timer_reset.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A72x1_A53x1.pctl.timer_reset.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A72x1_A53x1.pctl.timer_reset.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

## 13.46 FVP_Base_Cortex-A72x2

List of instances in FVP_Base_Cortex-A72x2.

### FVP_Base_Cortex-A72x2 instances

**FVP_Base_Cortex_A72x2**

Base Platform Compute Subsystem for ARMCortexA72x2CT.
Type: `FVP_Base_Cortex_A72x2`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A72x2.bp**

Peripherals and address map for the Base Platform.
Type: `BasePlatformPeripherals`.

**FVP_Base_Cortex_A72x2.bp.Timer_0_1**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Cortex_A72x2.bp.Timer_0_1.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x2.bp.Timer_0_1.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x2.bp.Timer_0_1.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A72x2.bp.Timer_0_1.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A72x2.bp.Timer_2_3**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Cortex_A72x2.bp.Timer_2_3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x2.bp.Timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x2.bp.Timer_2_3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A72x2.bp.Timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A72x2.bp.ap_refclk**

ARM Generic Timer.
Type: `MemoryMappedGenericTimer`.

**FVP_Base_Cortex_A72x2.bp.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**FVP_Base_Cortex_A72x2.bp.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x2.bp.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x2.bp.clock300MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x2.bp.clock32KHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x2.bp.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x2.bp.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x2.bp.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x2.bp.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x2.bp.dmc**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A72x2.bp.dmc_phy**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A72x2.bp.dummy_local_dap_rom**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A72x2.bp.dummy_ram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A72x2.bp.dummy_usb**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A72x2.bp.exclusive_monitor**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_Base_Cortex_A72x2.bp.flash0**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A72x2.bp.flash1**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A72x2.bp.flashloader0**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A72x2.bp.flashloader1**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A72x2.bp.generic_watchdog**

ARM Generic Watchdog.
Type: `MemoryMappedGenericWatchdog`.

**FVP_Base_Cortex_A72x2.bp.hdlcd0**

ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).
Type: `PL370_HDLCD`.

**FVP_Base_Cortex_A72x2.bp.hdlcd0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A72x2.bp.hdlcd0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A72x2.bp.hdlcd0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A72x2.bp.hdlcd0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A72x2.bp.hdlcd0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A72x2.bp.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_Base_Cortex_A72x2.bp.mmc**

Generic Multimedia Card.
Type: `MMC`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A72x2.bp.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A72x2.bp.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A72x2.bp.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A72x2.bp.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A72x2.bp.mpe**

Encrypt memory transactions for each encryption context with an independent key to prevent mismatch access.
Type: `PVMemoryProtectionEngine`.

**FVP_Base_Cortex_A72x2.bp.nontrustedrom**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A72x2.bp.nontrustedromloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A72x2.bp.ns_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A72x2.bp.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A72x2.bp.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x2.bp.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A72x2.bp.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A72x2.bp.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A72x2.bp.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A72x2.bp.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A72x2.bp.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x2.bp.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ClockTimerThread`.

**FVP_Base_Cortex_A72x2.bp.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A72x2.bp.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A72x2.bp.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A72x2.bp.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A72x2.bp.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x2.bp.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A72x2.bp.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A72x2.bp.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A72x2.bp.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A72x2.bp.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A72x2.bp.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x2.bp.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A72x2.bp.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A72x2.bp.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A72x2.bp.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A72x2.bp.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A72x2.bp.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A72x2.bp.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A72x2.bp.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A72x2.bp.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A72x2.bp.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_Base_Cortex_A72x2.bp.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A72x2.bp.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A72x2.bp.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A72x2.bp.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A72x2.bp.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A72x2.bp.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x2.bp.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A72x2.bp.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x2.bp.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_Base_Cortex_A72x2.bp.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

**FVP_Base_Cortex_A72x2.bp.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A72x2.bp.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A72x2.bp.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A72x2.bp.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A72x2.bp.pl111_clcd_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A72x2.bp.pl180_mci**

ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

**FVP_Base_Cortex_A72x2.bp.ps2keyboard**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

**FVP_Base_Cortex_A72x2.bp.ps2keyboard.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A72x2.bp.ps2keyboard.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A72x2.bp.ps2keyboard.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A72x2.bp.ps2keyboard.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A72x2.bp.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

**FVP_Base_Cortex_A72x2.bp.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A72x2.bp.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A72x2.bp.ps2mouse.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A72x2.bp.ps2mouse.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A72x2.bp.psram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A72x2.bp.refcounter**

Memory Mapped Counter Module for Generic Timers.
Type: `MemoryMappedCounterModule`.

**FVP_Base_Cortex_A72x2.bp.reset_or**

Or Gate.
Type: `OrGate`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A72x2.bp.rl_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A72x2.bp.rt_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A72x2.bp.s_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A72x2.bp.secureDRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A72x2.bp.secureSRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A72x2.bp.secureflash**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A72x2.bp.secureflashloader**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A72x2.bp.smsc_91c111**
SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_Base_Cortex_A72x2.bp.sp805_wdog**
ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A72x2.bp.sp805_wdog.clocktimer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A72x2.bp.sp805_wdog.clocktimer.timer**
A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A72x2.bp.sp805_wdog.clocktimer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A72x2.bp.sp805_wdog.clocktimer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A72x2.bp.sp810_sysctrl**
Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_Base_Cortex_A72x2.bp.sp810_sysctrl.clkdiv_clk0**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x2.bp.sp810_sysctrl.clkdiv_clk1**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x2.bp.sp810_sysctrl.clkdiv_clk2**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x2.bp.sp810_sysctrl.clkdiv_clk3**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x2.bp.sram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A72x2.bp.terminal_0**
Telnet terminal interface.
Type: `TelnetTerminal`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A72x2.bp.terminal_1**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A72x2.bp.terminal_2**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A72x2.bp.terminal_3**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A72x2.bp.trusted_key_storage**

Trusted Root-Key Storage unit.
Type: `RootKeyStorage`.

**FVP_Base_Cortex_A72x2.bp.trusted_nv_counter**

Trusted Non-Volatile Counter unit.
Type: `NonVolatileCounter`.

**FVP_Base_Cortex_A72x2.bp.trusted_rng**

Random Number Generator unit.
Type: `RandomNumberGenerator`.

**FVP_Base_Cortex_A72x2.bp.trusted_watchdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A72x2.bp.trusted_watchdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A72x2.bp.trusted_watchdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A72x2.bp.trusted_watchdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A72x2.bp.trusted_watchdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A72x2.bp.tzc_400**

TrustZone Address Space Controller.
Type: `TZC_400`.

**FVP_Base_Cortex_A72x2.bp.ve_sysregs**

Type: `VE_SysRegs`.

**FVP_Base_Cortex_A72x2.bp.vedcc**

Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

**FVP_Base_Cortex_A72x2.bp.virtio_net**

VirtioNet device over MMIO transport.
Type: `VirtioNetMMIO`.

**FVP_Base_Cortex_A72x2.bp.virtio_net_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A72x2.bp.virtio_rng**

VirtioEntropy device over MMIO transport.
Type: `VirtioEntropyMMIO`.

**FVP_Base_Cortex_A72x2.bp.virtioblockdevice**

VirtioBlock device over MMIO transport.
Type: `VirtioBlockMMIO`.

**FVP_Base_Cortex_A72x2.bp.virtioblockdevice_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A72x2.bp.virtiop9device**

virtio P9 server.
Type: `VirtioP9Device`.

**FVP_Base_Cortex_A72x2.bp.virtiop9device_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A72x2.bp.vis**

Display window for VE using Visualisation library.
Type: `VEVisualisation`.

**FVP_Base_Cortex_A72x2.bp.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_Base_Cortex_A72x2.bp.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x2.bp.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A72x2.bp.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A72x2.bp.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A72x2.bp.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A72x2.bp.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x2.bp.vram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A72x2.cci400**

Cache Coherent Interconnect for AXI4 ACE.
Type: `CCI400`.

**FVP_Base_Cortex_A72x2.clockdivider0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x2.cluster0**
ARM Cortex-A72 Cluster CT model.
Type: `Cluster_ARM_Cortex-A72`.

**FVP_Base_Cortex_A72x2.cluster0.cpu0**
ARM Cortex-A72 CT model.
Type: `ARM_Cortex-A72`.

**FVP_Base_Cortex_A72x2.cluster0.cpu0.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A72x2.cluster0.cpu0.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A72x2.cluster0.cpu0.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A72x2.cluster0.cpu1**
ARM Cortex-A72 CT model.
Type: `ARM_Cortex-A72`.

**FVP_Base_Cortex_A72x2.cluster0.cpu1.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A72x2.cluster0.cpu1.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A72x2.cluster0.cpu1.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A72x2.cluster0.l2_cache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A72x2.cluster0_labeller**
Type: `Labeller`.

**FVP_Base_Cortex_A72x2.dapmemlogger**
Bus Logger.
Type: `PVBusLogger`.

**FVP_Base_Cortex_A72x2.elfloader**
ELF loader component.
Type: `ElfLoader`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A72x2.gic_distributor**

GIC Interrupt Redistribution Infrastucture component.
Type: `GIC_IRI`.

**FVP_Base_Cortex_A72x2.pctl**

Base Platforms Power Controller.
Type: `Base_PowerController`.

**FVP_Base_Cortex_A72x2.pctl.timer_reset**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A72x2.pctl.timer_reset.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A72x2.pctl.timer_reset.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A72x2.pctl.timer_reset.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

## 13.47 FVP_Base_Cortex-A72x2-A53x4

List of instances in FVP_Base_Cortex-A72x2-A53x4.

### FVP_Base_Cortex-A72x2-A53x4 instances

**FVP_Base_Cortex_A72x2_A53x4**

Base Platform Compute Subsystem for ARMCortexA72x2CT and ARMCortexA53x4CT.
Type: `FVP_Base_Cortex_A72x2_A53x4`.

**FVP_Base_Cortex_A72x2_A53x4.bp**

Peripherals and address map for the Base Platform.
Type: `BasePlatformPeripherals`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A72x2_A53x4.bp.Timer_0_1**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Cortex_A72x2_A53x4.bp.Timer_0_1.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x2_A53x4.bp.Timer_0_1.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x2_A53x4.bp.Timer_0_1.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A72x2_A53x4.bp.Timer_0_1.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A72x2_A53x4.bp.Timer_2_3**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Cortex_A72x2_A53x4.bp.Timer_2_3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x2_A53x4.bp.Timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x2_A53x4.bp.Timer_2_3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A72x2_A53x4.bp.Timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A72x2_A53x4.bp.ap_refclk**

ARM Generic Timer.
Type: `MemoryMappedGenericTimer`.

**FVP_Base_Cortex_A72x2_A53x4.bp.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**FVP_Base_Cortex_A72x2_A53x4.bp.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x2_A53x4.bp.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x2_A53x4.bp.clock300MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x2_A53x4.bp.clock32KHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x2_A53x4.bp.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x2_A53x4.bp.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A72x2_A53x4.bp.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x2_A53x4.bp.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x2_A53x4.bp.dmc**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A72x2_A53x4.bp.dmc_phy**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A72x2_A53x4.bp.dummy_local_dap_rom**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A72x2_A53x4.bp.dummy_ram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A72x2_A53x4.bp.dummy_usb**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A72x2_A53x4.bp.exclusive_monitor**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_Base_Cortex_A72x2_A53x4.bp.flash0**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A72x2_A53x4.bp.flash1**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A72x2_A53x4.bp.flashloader0**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A72x2_A53x4.bp.flashloader1**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A72x2_A53x4.bp.generic_watchdog**

ARM Generic Watchdog.
Type: `MemoryMappedGenericWatchdog`.

**FVP_Base_Cortex_A72x2_A53x4.bp.hdlcd0**

ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).
Type: `PL370_HDLCD`.

**FVP_Base_Cortex_A72x2_A53x4.bp.hdlcd0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A72x2_A53x4.bp.hdlcd0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A72x2_A53x4.bp.hdlcd0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A72x2_A53x4.bp.hdlcd0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A72x2_A53x4.bp.hdlcd0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A72x2_A53x4.bp.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_Base_Cortex_A72x2_A53x4.bp.mmc**

Generic Multimedia Card.
Type: `MMC`.

**FVP_Base_Cortex_A72x2_A53x4.bp.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A72x2_A53x4.bp.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A72x2_A53x4.bp.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A72x2_A53x4.bp.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A72x2_A53x4.bp.mpe**

Encrypt memory transactions for each encryption context with an independent key to prevent mismatch access.
Type: `PVMemoryProtectionEngine`.

**FVP_Base_Cortex_A72x2_A53x4.bp.nontrustedrom**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A72x2_A53x4.bp.nontrustedromloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A72x2_A53x4.bp.ns_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A72x2_A53x4.bp.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A72x2_A53x4.bp.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A72x2_A53x4.bp.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A72x2_A53x4.bp.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A72x2_A53x4.bp.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A72x2_A53x4.bp.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A72x2_A53x4.bp.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A72x2_A53x4.bp.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x2_A53x4.bp.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A72x2_A53x4.bp.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A72x2_A53x4.bp.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A72x2_A53x4.bp.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A72x2_A53x4.bp.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A72x2_A53x4.bp.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x2_A53x4.bp.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A72x2_A53x4.bp.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A72x2_A53x4.bp.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A72x2_A53x4.bp.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A72x2_A53x4.bp.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A72x2_A53x4.bp.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x2_A53x4.bp.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A72x2_A53x4.bp.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A72x2_A53x4.bp.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A72x2_A53x4.bp.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A72x2_A53x4.bp.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

**FVP_Base_Cortex_A72x2_A53x4.bp.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal()

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A72x2_A53x4.bp.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A72x2_A53x4.bp.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A72x2_A53x4.bp.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A72x2_A53x4.bp.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_Base_Cortex_A72x2_A53x4.bp.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A72x2_A53x4.bp.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A72x2_A53x4.bp.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A72x2_A53x4.bp.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A72x2_A53x4.bp.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A72x2_A53x4.bp.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x2_A53x4.bp.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A72x2_A53x4.bp.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x2_A53x4.bp.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_Base_Cortex_A72x2_A53x4.bp.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

**FVP_Base_Cortex_A72x2_A53x4.bp.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A72x2_A53x4.bp.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A72x2_A53x4.bp.pl111_clcd.pl11x_clcd.timer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A72x2_A53x4.bp.pl111_clcd.pl11x_clcd.timer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A72x2_A53x4.bp.pl111_clcd_labeller**
Type: `Labeller`.

**FVP_Base_Cortex_A72x2_A53x4.bp.pl180_mci**
ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

**FVP_Base_Cortex_A72x2_A53x4.bp.ps2keyboard**
Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

**FVP_Base_Cortex_A72x2_A53x4.bp.ps2keyboard.ps2_clocktimer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A72x2_A53x4.bp.ps2keyboard.ps2_clocktimer.timer**
A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A72x2_A53x4.bp.ps2keyboard.ps2_clocktimer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A72x2_A53x4.bp.ps2keyboard.ps2_clocktimer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A72x2_A53x4.bp.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

**FVP_Base_Cortex_A72x2_A53x4.bp.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A72x2_A53x4.bp.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A72x2_A53x4.bp.ps2mouse.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A72x2_A53x4.bp.ps2mouse.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A72x2_A53x4.bp.psram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A72x2_A53x4.bp.refcounter**

Memory Mapped Counter Module for Generic Timers.
Type: `MemoryMappedCounterModule`.

**FVP_Base_Cortex_A72x2_A53x4.bp.reset_or**

Or Gate.
Type: `OrGate`.

**FVP_Base_Cortex_A72x2_A53x4.bp.rl_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A72x2_A53x4.bp.rt_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A72x2_A53x4.bp.s_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A72x2_A53x4.bp.secureDRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A72x2_A53x4.bp.secureSRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A72x2_A53x4.bp.secureflash**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A72x2_A53x4.bp.secureflashloader**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A72x2_A53x4.bp.smsc_91c111**
SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_Base_Cortex_A72x2_A53x4.bp.sp805_wdog**
ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A72x2_A53x4.bp.sp805_wdog.clocktimer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A72x2_A53x4.bp.sp805_wdog.clocktimer.timer**
A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A72x2_A53x4.bp.sp805_wdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A72x2_A53x4.bp.sp805_wdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A72x2_A53x4.bp.sp810_sysctrl**

Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_Base_Cortex_A72x2_A53x4.bp.sp810_sysctrl.clkdiv_clk0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x2_A53x4.bp.sp810_sysctrl.clkdiv_clk1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x2_A53x4.bp.sp810_sysctrl.clkdiv_clk2**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x2_A53x4.bp.sp810_sysctrl.clkdiv_clk3**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x2_A53x4.bp.sram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A72x2_A53x4.bp.terminal_0**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A72x2_A53x4.bp.terminal_1**

Telnet terminal interface.
Type: `TelnetTerminal`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A72x2_A53x4.bp.terminal_2**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A72x2_A53x4.bp.terminal_3**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A72x2_A53x4.bp.trusted_key_storage**

Trusted Root-Key Storage unit.
Type: `RootKeyStorage`.

**FVP_Base_Cortex_A72x2_A53x4.bp.trusted_nv_counter**

Trusted Non-Volatile Counter unit.
Type: `NonVolatileCounter`.

**FVP_Base_Cortex_A72x2_A53x4.bp.trusted_rng**

Random Number Generator unit.
Type: `RandomNumberGenerator`.

**FVP_Base_Cortex_A72x2_A53x4.bp.trusted_watchdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A72x2_A53x4.bp.trusted_watchdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A72x2_A53x4.bp.trusted_watchdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A72x2_A53x4.bp.trusted_watchdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A72x2_A53x4.bp.trusted_watchdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A72x2_A53x4.bp.tzc_400**

TrustZone Address Space Controller.
Type: `TZC_400`.

**FVP_Base_Cortex_A72x2_A53x4.bp.ve_sysregs**

Type: `VE_SysRegs`.

**FVP_Base_Cortex_A72x2_A53x4.bp.vedcc**

Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

**FVP_Base_Cortex_A72x2_A53x4.bp.virtio_net**

VirtioNet device over MMIO transport.
Type: `VirtioNetMMIO`.

**FVP_Base_Cortex_A72x2_A53x4.bp.virtio_net_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A72x2_A53x4.bp.virtio_rng**

VirtioEntropy device over MMIO transport.
Type: `VirtioEntropyMMIO`.

**FVP_Base_Cortex_A72x2_A53x4.bp.virtioblockdevice**

VirtioBlock device over MMIO transport.
Type: `VirtioBlockMMIO`.

**FVP_Base_Cortex_A72x2_A53x4.bp.virtioblockdevice_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A72x2_A53x4.bp.virtiop9device**

virtio P9 server.
Type: `VirtioP9Device`.

**FVP_Base_Cortex_A72x2_A53x4.bp.virtiop9device_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A72x2_A53x4.bp.vis**

Display window for VE using Visualisation library.
Type: `VEVisualisation`.

**FVP_Base_Cortex_A72x2_A53x4.bp.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_Base_Cortex_A72x2_A53x4.bp.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A72x2_A53x4.bp.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A72x2_A53x4.bp.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A72x2_A53x4.bp.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A72x2_A53x4.bp.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A72x2_A53x4.bp.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x2_A53x4.bp.vram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A72x2_A53x4.cci400**

Cache Coherent Interconnect for AXI4 ACE.
Type: `CCI400`.

**FVP_Base_Cortex_A72x2_A53x4.clockdivider0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A72x2_A53x4.clockdivider1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x2_A53x4.cluster0**

ARM Cortex-A72 Cluster CT model.
Type: `Cluster_ARM_Cortex-A72`.

**FVP_Base_Cortex_A72x2_A53x4.cluster0.cpu0**

ARM Cortex-A72 CT model.
Type: `ARM_Cortex-A72`.

**FVP_Base_Cortex_A72x2_A53x4.cluster0.cpu0.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A72x2_A53x4.cluster0.cpu0.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A72x2_A53x4.cluster0.cpu0.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A72x2_A53x4.cluster0.cpu1**

ARM Cortex-A72 CT model.
Type: `ARM_Cortex-A72`.

**FVP_Base_Cortex_A72x2_A53x4.cluster0.cpu1.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A72x2_A53x4.cluster0.cpu1.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A72x2_A53x4.cluster0.cpu1.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A72x2_A53x4.cluster0.l2_cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A72x2_A53x4.cluster0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A72x2_A53x4.cluster1**

ARM Cortex-A53 Cluster CT model.
Type: `Cluster_ARM_Cortex-A53`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A72x2_A53x4.cluster1.cpu0**

ARM Cortex-A53 CT model.
Type: `ARM_Cortex-A53`.

**FVP_Base_Cortex_A72x2_A53x4.cluster1.cpu0.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A72x2_A53x4.cluster1.cpu0.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A72x2_A53x4.cluster1.cpu0.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A72x2_A53x4.cluster1.cpu1**

ARM Cortex-A53 CT model.
Type: `ARM_Cortex-A53`.

**FVP_Base_Cortex_A72x2_A53x4.cluster1.cpu1.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A72x2_A53x4.cluster1.cpu1.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A72x2_A53x4.cluster1.cpu1.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A72x2_A53x4.cluster1.cpu2**

ARM Cortex-A53 CT model.
Type: `ARM_Cortex-A53`.

**FVP_Base_Cortex_A72x2_A53x4.cluster1.cpu2.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A72x2_A53x4.cluster1.cpu2.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A72x2_A53x4.cluster1.cpu2.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A72x2_A53x4.cluster1.cpu3**

ARM Cortex-A53 CT model.
Type: `ARM_Cortex-A53`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A72x2_A53x4.cluster1.cpu3.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A72x2_A53x4.cluster1.cpu3.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A72x2_A53x4.cluster1.cpu3.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A72x2_A53x4.cluster1.l2_cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A72x2_A53x4.cluster1_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A72x2_A53x4.dapmemlogger**

Bus Logger.
Type: `PVBusLogger`.

**FVP_Base_Cortex_A72x2_A53x4.elfloader**

ELF loader component.
Type: `ElfLoader`.

**FVP_Base_Cortex_A72x2_A53x4.gic_distributor**

GIC Interrupt Redistribution Infrastucture component.
Type: `GIC_IRI`.

**FVP_Base_Cortex_A72x2_A53x4.pctl**

Base Platforms Power Controller.
Type: `Base_PowerController`.

**FVP_Base_Cortex_A72x2_A53x4.pctl.timer_reset**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A72x2_A53x4.pctl.timer_reset.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A72x2_A53x4.pctl.timer_reset.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A72x2_A53x4.pctl.timer_reset.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

## 13.48 FVP_Base_Cortex-A72x4

List of instances in FVP_Base_Cortex-A72x4.

### FVP_Base_Cortex-A72x4 instances

**FVP_Base_Cortex_A72x4**

Base Platform Compute Subsystem for ARMCortexA72x4CT.
Type: `FVP_Base_Cortex_A72x4`.

**FVP_Base_Cortex_A72x4.bp**

Peripherals and address map for the Base Platform.
Type: `BasePlatformPeripherals`.

**FVP_Base_Cortex_A72x4.bp.Timer_0_1**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Cortex_A72x4.bp.Timer_0_1.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x4.bp.Timer_0_1.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x4.bp.Timer_0_1.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A72x4.bp.Timer_0_1.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A72x4.bp.Timer_2_3**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Cortex_A72x4.bp.Timer_2_3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x4.bp.Timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x4.bp.Timer_2_3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A72x4.bp.Timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A72x4.bp.ap_refclk**

ARM Generic Timer.
Type: `MemoryMappedGenericTimer`.

**FVP_Base_Cortex_A72x4.bp.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**FVP_Base_Cortex_A72x4.bp.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x4.bp.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x4.bp.clock300MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x4.bp.clock32KHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x4.bp.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x4.bp.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x4.bp.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x4.bp.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x4.bp.dmc**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A72x4.bp.dmc_phy**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A72x4.bp.dummy_local_dap_rom**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A72x4.bp.dummy_ram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A72x4.bp.dummy_usb**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A72x4.bp.exclusive_monitor**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_Base_Cortex_A72x4.bp.flash0**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A72x4.bp.flash1**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A72x4.bp.flashloader0**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A72x4.bp.flashloader1**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A72x4.bp.generic_watchdog**

ARM Generic Watchdog.
Type: `MemoryMappedGenericWatchdog`.

**FVP_Base_Cortex_A72x4.bp.hdlcd0**

ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).
Type: `PL370_HDLCD`.

**FVP_Base_Cortex_A72x4.bp.hdlcd0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A72x4.bp.hdlcd0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A72x4.bp.hdlcd0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A72x4.bp.hdlcd0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A72x4.bp.hdlcd0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A72x4.bp.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_Base_Cortex_A72x4.bp.mmc**

Generic Multimedia Card.
Type: `MMC`.

**FVP_Base_Cortex_A72x4.bp.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A72x4.bp.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A72x4.bp.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A72x4.bp.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A72x4.bp.mpe**

Encrypt memory transactions for each encryption context with an independent key to prevent mismatch access.
Type: `PVMemoryProtectionEngine`.

**FVP_Base_Cortex_A72x4.bp.nontrustedrom**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A72x4.bp.nontrustedromloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A72x4.bp.ns_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A72x4.bp.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A72x4.bp.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x4.bp.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A72x4.bp.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A72x4.bp.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A72x4.bp.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A72x4.bp.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A72x4.bp.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x4.bp.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A72x4.bp.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A72x4.bp.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A72x4.bp.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A72x4.bp.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A72x4.bp.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x4.bp.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A72x4.bp.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A72x4.bp.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A72x4.bp.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A72x4.bp.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A72x4.bp.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x4.bp.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ClockTimerThread`.

**FVP_Base_Cortex_A72x4.bp.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A72x4.bp.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A72x4.bp.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A72x4.bp.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

**FVP_Base_Cortex_A72x4.bp.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A72x4.bp.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A72x4.bp.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A72x4.bp.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A72x4.bp.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_Base_Cortex_A72x4.bp.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A72x4.bp.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A72x4.bp.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A72x4.bp.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A72x4.bp.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A72x4.bp.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x4.bp.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A72x4.bp.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x4.bp.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_Base_Cortex_A72x4.bp.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

**FVP_Base_Cortex_A72x4.bp.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A72x4.bp.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A72x4.bp.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A72x4.bp.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A72x4.bp.pl111_clcd_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A72x4.bp.pl180_mci**

ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

**FVP_Base_Cortex_A72x4.bp.ps2keyboard**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A72x4.bp.ps2keyboard.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A72x4.bp.ps2keyboard.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A72x4.bp.ps2keyboard.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A72x4.bp.ps2keyboard.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A72x4.bp.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

**FVP_Base_Cortex_A72x4.bp.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A72x4.bp.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A72x4.bp.ps2mouse.ps2_clocktimer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A72x4.bp.ps2mouse.ps2_clocktimer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A72x4.bp.psram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A72x4.bp.refcounter**
Memory Mapped Counter Module for Generic Timers.
Type: `MemoryMappedCounterModule`.

**FVP_Base_Cortex_A72x4.bp.reset_or**
Or Gate.
Type: `OrGate`.

**FVP_Base_Cortex_A72x4.bp.rl_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A72x4.bp.rt_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A72x4.bp.s_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A72x4.bp.secureDRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A72x4.bp.secureSRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A72x4.bp.secureflash**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A72x4.bp.secureflashloader**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A72x4.bp.smsc_91c111**

SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_Base_Cortex_A72x4.bp.sp805_wdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A72x4.bp.sp805_wdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A72x4.bp.sp805_wdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A72x4.bp.sp805_wdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A72x4.bp.sp805_wdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A72x4.bp.sp810_sysctrl**

Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_Base_Cortex_A72x4.bp.sp810_sysctrl.clkdiv_clk0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x4.bp.sp810_sysctrl.clkdiv_clk1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x4.bp.sp810_sysctrl.clkdiv_clk2**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x4.bp.sp810_sysctrl.clkdiv_clk3**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x4.bp.sram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A72x4.bp.terminal_0**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A72x4.bp.terminal_1**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A72x4.bp.terminal_2**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A72x4.bp.terminal_3**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A72x4.bp.trusted_key_storage**
Trusted Root-Key Storage unit.
Type: `RootKeyStorage`.

**FVP_Base_Cortex_A72x4.bp.trusted_nv_counter**
Trusted Non-Volatile Counter unit.
Type: `NonVolatileCounter`.

**FVP_Base_Cortex_A72x4.bp.trusted_rng**
Random Number Generator unit.
Type: `RandomNumberGenerator`.

**FVP_Base_Cortex_A72x4.bp.trusted_watchdog**
ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A72x4.bp.trusted_watchdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A72x4.bp.trusted_watchdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A72x4.bp.trusted_watchdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A72x4.bp.trusted_watchdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A72x4.bp.tzc_400**

TrustZone Address Space Controller.
Type: `TZC_400`.

**FVP_Base_Cortex_A72x4.bp.ve_sysregs**

Type: `VE_SysRegs`.

**FVP_Base_Cortex_A72x4.bp.vedcc**

Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

**FVP_Base_Cortex_A72x4.bp.virtio_net**

VirtioNet device over MMIO transport.
Type: `VirtioNetMMIO`.

**FVP_Base_Cortex_A72x4.bp.virtio_net_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A72x4.bp.virtio_rng**

VirtioEntropy device over MMIO transport.
Type: `VirtioEntropyMMIO`.

**FVP_Base_Cortex_A72x4.bp.virtioblockdevice**

VirtioBlock device over MMIO transport.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `VirtioBlockMMIO`.

**FVP_Base_Cortex_A72x4.bp.virtioblockdevice_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A72x4.bp.virtiop9device**

virtio P9 server.
Type: `VirtioP9Device`.

**FVP_Base_Cortex_A72x4.bp.virtiop9device_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A72x4.bp.vis**

Display window for VE using Visualisation library.
Type: `VEVisualisation`.

**FVP_Base_Cortex_A72x4.bp.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_Base_Cortex_A72x4.bp.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x4.bp.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A72x4.bp.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A72x4.bp.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A72x4.bp.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A72x4.bp.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x4.bp.vram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A72x4.cci400**

Cache Coherent Interconnect for AXI4 ACE.
Type: `CCI400`.

**FVP_Base_Cortex_A72x4.clockdivider0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x4.cluster0**

ARM Cortex-A72 Cluster CT model.
Type: `Cluster_ARM_Cortex-A72`.

**FVP_Base_Cortex_A72x4.cluster0.cpu0**

ARM Cortex-A72 CT model.
Type: `ARM_Cortex-A72`.

**FVP_Base_Cortex_A72x4.cluster0.cpu0.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A72x4.cluster0.cpu0.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A72x4.cluster0.cpu0.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A72x4.cluster0.cpu1**

ARM Cortex-A72 CT model.
Type: `ARM_Cortex-A72`.

**FVP_Base_Cortex_A72x4.cluster0.cpu1.dtlb**

TLB - instruction, data or unified.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `TlbCadi`.

**FVP_Base_Cortex_A72x4.cluster0.cpu1.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A72x4.cluster0.cpu1.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A72x4.cluster0.cpu2**

ARM Cortex-A72 CT model.
Type: `ARM_Cortex-A72`.

**FVP_Base_Cortex_A72x4.cluster0.cpu2.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A72x4.cluster0.cpu2.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A72x4.cluster0.cpu2.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A72x4.cluster0.cpu3**

ARM Cortex-A72 CT model.
Type: `ARM_Cortex-A72`.

**FVP_Base_Cortex_A72x4.cluster0.cpu3.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A72x4.cluster0.cpu3.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A72x4.cluster0.cpu3.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A72x4.cluster0.l2_cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A72x4.cluster0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A72x4.dapmemlogger**

Bus Logger.
Type: `PVBusLogger`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A72x4.elfloader**

ELF loader component.
Type: `ElfLoader`.

**FVP_Base_Cortex_A72x4.gic_distributor**

GIC Interrupt Redistribution Infrastucture component.
Type: `GIC_IRI`.

**FVP_Base_Cortex_A72x4.pctl**

Base Platforms Power Controller.
Type: `Base_PowerController`.

**FVP_Base_Cortex_A72x4.pctl.timer_reset**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A72x4.pctl.timer_reset.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A72x4.pctl.timer_reset.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A72x4.pctl.timer_reset.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

## 13.49 FVP_Base_Cortex-A72x4-A53x4

List of instances in FVP_Base_Cortex-A72x4-A53x4.

### FVP_Base_Cortex-A72x4-A53x4 instances

**FVP_Base_Cortex_A72x4_A53x4**

Base Platform Compute Subsystem for ARMCortexA72x4CT and ARMCortexA53x4CT.
Type: `FVP_Base_Cortex_A72x4_A53x4`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A72x4_A53x4.bp**

Peripherals and address map for the Base Platform.
Type: `BasePlatformPeripherals`.

**FVP_Base_Cortex_A72x4_A53x4.bp.Timer_0_1**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Cortex_A72x4_A53x4.bp.Timer_0_1.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x4_A53x4.bp.Timer_0_1.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x4_A53x4.bp.Timer_0_1.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A72x4_A53x4.bp.Timer_0_1.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A72x4_A53x4.bp.Timer_2_3**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Cortex_A72x4_A53x4.bp.Timer_2_3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x4_A53x4.bp.Timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x4_A53x4.bp.Timer_2_3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A72x4_A53x4.bp.Timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A72x4_A53x4.bp.ap_refclk**

ARM Generic Timer.
Type: `MemoryMappedGenericTimer`.

**FVP_Base_Cortex_A72x4_A53x4.bp.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**FVP_Base_Cortex_A72x4_A53x4.bp.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x4_A53x4.bp.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x4_A53x4.bp.clock300MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x4_A53x4.bp.clock32KHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x4_A53x4.bp.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x4_A53x4.bp.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x4_A53x4.bp.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x4_A53x4.bp.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x4_A53x4.bp.dmc**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A72x4_A53x4.bp.dmc_phy**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A72x4_A53x4.bp.dummy_local_dap_rom**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A72x4_A53x4.bp.dummy_ram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A72x4_A53x4.bp.dummy_usb**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A72x4_A53x4.bp.exclusive_monitor**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_Base_Cortex_A72x4_A53x4.bp.flash0**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A72x4_A53x4.bp.flash1**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A72x4_A53x4.bp.flashloader0**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A72x4_A53x4.bp.flashloader1**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A72x4_A53x4.bp.generic_watchdog**

ARM Generic Watchdog.
Type: `MemoryMappedGenericWatchdog`.

**FVP_Base_Cortex_A72x4_A53x4.bp.hdlcd0**

ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).
Type: `PL370_HDLCD`.

**FVP_Base_Cortex_A72x4_A53x4.bp.hdlcd0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A72x4_A53x4.bp.hdlcd0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A72x4_A53x4.bp.hdlcd0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A72x4_A53x4.bp.hdlcd0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A72x4_A53x4.bp.hdlcd0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A72x4_A53x4.bp.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_Base_Cortex_A72x4_A53x4.bp.mmc**

Generic Multimedia Card.
Type: `MMC`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A72x4_A53x4.bp.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A72x4_A53x4.bp.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A72x4_A53x4.bp.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A72x4_A53x4.bp.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A72x4_A53x4.bp.mpe**

Encrypt memory transactions for each encryption context with an independent key to prevent mismatch access.
Type: `PVMemoryProtectionEngine`.

**FVP_Base_Cortex_A72x4_A53x4.bp.nontrustedrom**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A72x4_A53x4.bp.nontrustedromloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A72x4_A53x4.bp.ns_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A72x4_A53x4.bp.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A72x4_A53x4.bp.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x4_A53x4.bp.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A72x4_A53x4.bp.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A72x4_A53x4.bp.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A72x4_A53x4.bp.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A72x4_A53x4.bp.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A72x4_A53x4.bp.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x4_A53x4.bp.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ClockTimerThread`.

**FVP_Base_Cortex_A72x4_A53x4.bp.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A72x4_A53x4.bp.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A72x4_A53x4.bp.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A72x4_A53x4.bp.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A72x4_A53x4.bp.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x4_A53x4.bp.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A72x4_A53x4.bp.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A72x4_A53x4.bp.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A72x4_A53x4.bp.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A72x4_A53x4.bp.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A72x4_A53x4.bp.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x4_A53x4.bp.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A72x4_A53x4.bp.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A72x4_A53x4.bp.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A72x4_A53x4.bp.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A72x4_A53x4.bp.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A72x4_A53x4.bp.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A72x4_A53x4.bp.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A72x4_A53x4.bp.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A72x4_A53x4.bp.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A72x4_A53x4.bp.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_Base_Cortex_A72x4_A53x4.bp.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A72x4_A53x4.bp.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A72x4_A53x4.bp.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A72x4_A53x4.bp.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A72x4_A53x4.bp.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A72x4_A53x4.bp.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x4_A53x4.bp.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A72x4_A53x4.bp.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x4_A53x4.bp.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_Base_Cortex_A72x4_A53x4.bp.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

**FVP_Base_Cortex_A72x4_A53x4.bp.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A72x4_A53x4.bp.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A72x4_A53x4.bp.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A72x4_A53x4.bp.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A72x4_A53x4.bp.pl111_clcd_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A72x4_A53x4.bp.pl180_mci**

ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

**FVP_Base_Cortex_A72x4_A53x4.bp.ps2keyboard**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

**FVP_Base_Cortex_A72x4_A53x4.bp.ps2keyboard.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A72x4_A53x4.bp.ps2keyboard.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A72x4_A53x4.bp.ps2keyboard.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A72x4_A53x4.bp.ps2keyboard.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A72x4_A53x4.bp.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

**FVP_Base_Cortex_A72x4_A53x4.bp.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A72x4_A53x4.bp.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A72x4_A53x4.bp.ps2mouse.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A72x4_A53x4.bp.ps2mouse.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A72x4_A53x4.bp.psram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A72x4_A53x4.bp.refcounter**

Memory Mapped Counter Module for Generic Timers.
Type: `MemoryMappedCounterModule`.

**FVP_Base_Cortex_A72x4_A53x4.bp.reset_or**

Or Gate.
Type: `OrGate`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A72x4_A53x4.bp.rl_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A72x4_A53x4.bp.rt_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A72x4_A53x4.bp.s_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A72x4_A53x4.bp.secureDRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A72x4_A53x4.bp.secureSRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A72x4_A53x4.bp.secureflash**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A72x4_A53x4.bp.secureflashloader**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A72x4_A53x4.bp.smsc_91c111**
SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_Base_Cortex_A72x4_A53x4.bp.sp805_wdog**
ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A72x4_A53x4.bp.sp805_wdog.clocktimer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A72x4_A53x4.bp.sp805_wdog.clocktimer.timer**
A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A72x4_A53x4.bp.sp805_wdog.clocktimer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A72x4_A53x4.bp.sp805_wdog.clocktimer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A72x4_A53x4.bp.sp810_sysctrl**
Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_Base_Cortex_A72x4_A53x4.bp.sp810_sysctrl.clkdiv_clk0**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x4_A53x4.bp.sp810_sysctrl.clkdiv_clk1**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x4_A53x4.bp.sp810_sysctrl.clkdiv_clk2**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x4_A53x4.bp.sp810_sysctrl.clkdiv_clk3**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x4_A53x4.bp.sram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A72x4_A53x4.bp.terminal_0**
Telnet terminal interface.
Type: `TelnetTerminal`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A72x4_A53x4.bp.terminal_1**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A72x4_A53x4.bp.terminal_2**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A72x4_A53x4.bp.terminal_3**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A72x4_A53x4.bp.trusted_key_storage**

Trusted Root-Key Storage unit.
Type: `RootKeyStorage`.

**FVP_Base_Cortex_A72x4_A53x4.bp.trusted_nv_counter**

Trusted Non-Volatile Counter unit.
Type: `NonVolatileCounter`.

**FVP_Base_Cortex_A72x4_A53x4.bp.trusted_rng**

Random Number Generator unit.
Type: `RandomNumberGenerator`.

**FVP_Base_Cortex_A72x4_A53x4.bp.trusted_watchdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A72x4_A53x4.bp.trusted_watchdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A72x4_A53x4.bp.trusted_watchdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A72x4_A53x4.bp.trusted_watchdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A72x4_A53x4.bp.trusted_watchdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A72x4_A53x4.bp.tzc_400**

TrustZone Address Space Controller.
Type: `TZC_400`.

**FVP_Base_Cortex_A72x4_A53x4.bp.ve_sysregs**

Type: `VE_SysRegs`.

**FVP_Base_Cortex_A72x4_A53x4.bp.vedcc**

Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

**FVP_Base_Cortex_A72x4_A53x4.bp.virtio_net**

VirtioNet device over MMIO transport.
Type: `VirtioNetMMIO`.

**FVP_Base_Cortex_A72x4_A53x4.bp.virtio_net_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A72x4_A53x4.bp.virtio_rng**

VirtioEntropy device over MMIO transport.
Type: `VirtioEntropyMMIO`.

**FVP_Base_Cortex_A72x4_A53x4.bp.virtioblockdevice**

VirtioBlock device over MMIO transport.
Type: `VirtioBlockMMIO`.

**FVP_Base_Cortex_A72x4_A53x4.bp.virtioblockdevice_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A72x4_A53x4.bp.virtiop9device**

virtio P9 server.
Type: `VirtioP9Device`.

**FVP_Base_Cortex_A72x4_A53x4.bp.virtiop9device_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A72x4_A53x4.bp.vis**

Display window for VE using Visualisation library.
Type: `VEVisualisation`.

**FVP_Base_Cortex_A72x4_A53x4.bp.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_Base_Cortex_A72x4_A53x4.bp.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x4_A53x4.bp.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A72x4_A53x4.bp.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A72x4_A53x4.bp.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A72x4_A53x4.bp.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A72x4_A53x4.bp.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x4_A53x4.bp.vram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A72x4_A53x4.cci400**

Cache Coherent Interconnect for AXI4 ACE.
Type: `CCI400`.

**FVP_Base_Cortex_A72x4_A53x4.clockdivider0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x4_A53x4.clockdivider1**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A72x4_A53x4.cluster0**
ARM Cortex-A72 Cluster CT model.
Type: `Cluster_ARM_Cortex-A72`.

**FVP_Base_Cortex_A72x4_A53x4.cluster0.cpu0**
ARM Cortex-A72 CT model.
Type: `ARM_Cortex-A72`.

**FVP_Base_Cortex_A72x4_A53x4.cluster0.cpu0.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A72x4_A53x4.cluster0.cpu0.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A72x4_A53x4.cluster0.cpu0.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A72x4_A53x4.cluster0.cpu1**
ARM Cortex-A72 CT model.
Type: `ARM_Cortex-A72`.

**FVP_Base_Cortex_A72x4_A53x4.cluster0.cpu1.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A72x4_A53x4.cluster0.cpu1.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A72x4_A53x4.cluster0.cpu1.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A72x4_A53x4.cluster0.cpu2**
ARM Cortex-A72 CT model.
Type: `ARM_Cortex-A72`.

**FVP_Base_Cortex_A72x4_A53x4.cluster0.cpu2.dtlb**
TLB - instruction, data or unified.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `TlbCadi`.

**FVP_Base_Cortex_A72x4_A53x4.cluster0.cpu2.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A72x4_A53x4.cluster0.cpu2.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A72x4_A53x4.cluster0.cpu3**

ARM Cortex-A72 CT model.
Type: `ARM_Cortex-A72`.

**FVP_Base_Cortex_A72x4_A53x4.cluster0.cpu3.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A72x4_A53x4.cluster0.cpu3.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A72x4_A53x4.cluster0.cpu3.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A72x4_A53x4.cluster0.l2_cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A72x4_A53x4.cluster0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A72x4_A53x4.cluster1**

ARM Cortex-A53 Cluster CT model.
Type: `Cluster_ARM_Cortex-A53`.

**FVP_Base_Cortex_A72x4_A53x4.cluster1.cpu0**

ARM Cortex-A53 CT model.
Type: `ARM_Cortex-A53`.

**FVP_Base_Cortex_A72x4_A53x4.cluster1.cpu0.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A72x4_A53x4.cluster1.cpu0.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A72x4_A53x4.cluster1.cpu0.l1icache**

PV Cache.
Type: `PVCache`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A72x4_A53x4.cluster1.cpu1**

ARM Cortex-A53 CT model.
Type: `ARM_Cortex-A53`.

**FVP_Base_Cortex_A72x4_A53x4.cluster1.cpu1.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A72x4_A53x4.cluster1.cpu1.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A72x4_A53x4.cluster1.cpu1.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A72x4_A53x4.cluster1.cpu2**

ARM Cortex-A53 CT model.
Type: `ARM_Cortex-A53`.

**FVP_Base_Cortex_A72x4_A53x4.cluster1.cpu2.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A72x4_A53x4.cluster1.cpu2.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A72x4_A53x4.cluster1.cpu2.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A72x4_A53x4.cluster1.cpu3**

ARM Cortex-A53 CT model.
Type: `ARM_Cortex-A53`.

**FVP_Base_Cortex_A72x4_A53x4.cluster1.cpu3.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A72x4_A53x4.cluster1.cpu3.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A72x4_A53x4.cluster1.cpu3.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A72x4_A53x4.cluster1.l2_cache**

PV Cache.
Type: `PVCache`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A72x4_A53x4.cluster1_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A72x4_A53x4.dapmemlogger**

Bus Logger.
Type: `PVBusLogger`.

**FVP_Base_Cortex_A72x4_A53x4.elfloader**

ELF loader component.
Type: `ElfLoader`.

**FVP_Base_Cortex_A72x4_A53x4.gic_distributor**

GIC Interrupt Redistribution Infrastucture component.
Type: `GIC_IRI`.

**FVP_Base_Cortex_A72x4_A53x4.pctl**

Base Platforms Power Controller.
Type: `Base_PowerController`.

**FVP_Base_Cortex_A72x4_A53x4.pctl.timer_reset**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A72x4_A53x4.pctl.timer_reset.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A72x4_A53x4.pctl.timer_reset.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A72x4_A53x4.pctl.timer_reset.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

# 13.50 FVP_Base_Cortex-A73x1

List of instances in FVP_Base_Cortex-A73x1.

## FVP_Base_Cortex-A73x1 instances

**FVP_Base_Cortex_A73x1**

Base Platform Compute Subsystem for ARMCortexA73x1CT.
Type: `FVP_Base_Cortex_A73x1`.

**FVP_Base_Cortex_A73x1.bp**

Peripherals and address map for the Base Platform.
Type: `BasePlatformPeripherals`.

**FVP_Base_Cortex_A73x1.bp.Timer_0_1**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Cortex_A73x1.bp.Timer_0_1.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x1.bp.Timer_0_1.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x1.bp.Timer_0_1.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A73x1.bp.Timer_0_1.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A73x1.bp.Timer_2_3**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Cortex_A73x1.bp.Timer_2_3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A73x1.bp.Timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x1.bp.Timer_2_3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A73x1.bp.Timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A73x1.bp.ap_refclk**

ARM Generic Timer.
Type: `MemoryMappedGenericTimer`.

**FVP_Base_Cortex_A73x1.bp.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**FVP_Base_Cortex_A73x1.bp.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x1.bp.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x1.bp.clock300MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x1.bp.clock32KHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A73x1.bp.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x1.bp.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x1.bp.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x1.bp.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x1.bp.dmc**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A73x1.bp.dmc_phy**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A73x1.bp.dummy_local_dap_rom**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A73x1.bp.dummy_ram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A73x1.bp.dummy_usb**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A73x1.bp.exclusive_monitor**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A73x1.bp.flash0**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A73x1.bp.flash1**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A73x1.bp.flashloader0**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A73x1.bp.flashloader1**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A73x1.bp.generic_watchdog**

ARM Generic Watchdog.
Type: `MemoryMappedGenericWatchdog`.

**FVP_Base_Cortex_A73x1.bp.hdlcd0**

ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).
Type: `PL370_HDLCD`.

**FVP_Base_Cortex_A73x1.bp.hdlcd0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A73x1.bp.hdlcd0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A73x1.bp.hdlcd0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A73x1.bp.hdlcd0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A73x1.bp.hdlcd0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A73x1.bp.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_Base_Cortex_A73x1.bp.mmc**

Generic Multimedia Card.
Type: `MMC`.

**FVP_Base_Cortex_A73x1.bp.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A73x1.bp.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A73x1.bp.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A73x1.bp.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A73x1.bp.mpe**

Encrypt memory transactions for each encryption context with an independent key to prevent mismatch access.
Type: `PVMemoryProtectionEngine`.

**FVP_Base_Cortex_A73x1.bp.nontrustedrom**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A73x1.bp.nontrustedromloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A73x1.bp.ns_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A73x1.bp.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A73x1.bp.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x1.bp.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A73x1.bp.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A73x1.bp.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A73x1.bp.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A73x1.bp.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A73x1.bp.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x1.bp.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A73x1.bp.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A73x1.bp.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A73x1.bp.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A73x1.bp.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A73x1.bp.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x1.bp.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ClockTimerThread`.

**FVP_Base_Cortex_A73x1.bp.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A73x1.bp.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A73x1.bp.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A73x1.bp.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A73x1.bp.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x1.bp.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A73x1.bp.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A73x1.bp.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A73x1.bp.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A73x1.bp.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

**FVP_Base_Cortex_A73x1.bp.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A73x1.bp.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A73x1.bp.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A73x1.bp.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A73x1.bp.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_Base_Cortex_A73x1.bp.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A73x1.bp.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A73x1.bp.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A73x1.bp.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A73x1.bp.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A73x1.bp.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x1.bp.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A73x1.bp.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x1.bp.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_Base_Cortex_A73x1.bp.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A73x1.bp.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A73x1.bp.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A73x1.bp.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A73x1.bp.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A73x1.bp.pl111_clcd_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A73x1.bp.pl180_mci**

ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

**FVP_Base_Cortex_A73x1.bp.ps2keyboard**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

**FVP_Base_Cortex_A73x1.bp.ps2keyboard.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A73x1.bp.ps2keyboard.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A73x1.bp.ps2keyboard.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A73x1.bp.ps2keyboard.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A73x1.bp.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

**FVP_Base_Cortex_A73x1.bp.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A73x1.bp.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A73x1.bp.ps2mouse.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A73x1.bp.ps2mouse.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A73x1.bp.psram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A73x1.bp.refcounter**
Memory Mapped Counter Module for Generic Timers.
Type: `MemoryMappedCounterModule`.

**FVP_Base_Cortex_A73x1.bp.reset_or**
Or Gate.
Type: `OrGate`.

**FVP_Base_Cortex_A73x1.bp.rl_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A73x1.bp.rt_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A73x1.bp.s_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A73x1.bp.secureDRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A73x1.bp.secureSRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A73x1.bp.secureflash**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A73x1.bp.secureflashloader**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A73x1.bp.smsc_91c111**
SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_Base_Cortex_A73x1.bp.sp805_wdog**
ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A73x1.bp.sp805_wdog.clocktimer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A73x1.bp.sp805_wdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A73x1.bp.sp805_wdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A73x1.bp.sp805_wdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A73x1.bp.sp810_sysctrl**

Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_Base_Cortex_A73x1.bp.sp810_sysctrl.clkdiv_clk0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x1.bp.sp810_sysctrl.clkdiv_clk1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x1.bp.sp810_sysctrl.clkdiv_clk2**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x1.bp.sp810_sysctrl.clkdiv_clk3**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**`FVP_Base_Cortex_A73x1.bp.sram`**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**`FVP_Base_Cortex_A73x1.bp.terminal_0`**
Telnet terminal interface.
Type: `TelnetTerminal`.

**`FVP_Base_Cortex_A73x1.bp.terminal_1`**
Telnet terminal interface.
Type: `TelnetTerminal`.

**`FVP_Base_Cortex_A73x1.bp.terminal_2`**
Telnet terminal interface.
Type: `TelnetTerminal`.

**`FVP_Base_Cortex_A73x1.bp.terminal_3`**
Telnet terminal interface.
Type: `TelnetTerminal`.

**`FVP_Base_Cortex_A73x1.bp.trusted_key_storage`**
Trusted Root-Key Storage unit.
Type: `RootKeyStorage`.

**`FVP_Base_Cortex_A73x1.bp.trusted_nv_counter`**
Trusted Non-Volatile Counter unit.
Type: `NonVolatileCounter`.

**`FVP_Base_Cortex_A73x1.bp.trusted_rng`**
Random Number Generator unit.
Type: `RandomNumberGenerator`.

**`FVP_Base_Cortex_A73x1.bp.trusted_watchdog`**
ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**`FVP_Base_Cortex_A73x1.bp.trusted_watchdog.clocktimer`**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**`FVP_Base_Cortex_A73x1.bp.trusted_watchdog.clocktimer.timer`**
A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A73x1.bp.trusted_watchdog.clocktimer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A73x1.bp.trusted_watchdog.clocktimer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A73x1.bp.tzc_400**
TrustZone Address Space Controller.
Type: `TZC_400`.

**FVP_Base_Cortex_A73x1.bp.ve_sysregs**
Type: `VE_SysRegs`.

**FVP_Base_Cortex_A73x1.bp.vedcc**
Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

**FVP_Base_Cortex_A73x1.bp.virtio_net**
VirtioNet device over MMIO transport.
Type: `VirtioNetMMIO`.

**FVP_Base_Cortex_A73x1.bp.virtio_net_labeller**
Type: `Labeller`.

**FVP_Base_Cortex_A73x1.bp.virtio_rng**
VirtioEntropy device over MMIO transport.
Type: `VirtioEntropyMMIO`.

**FVP_Base_Cortex_A73x1.bp.virtioblockdevice**
VirtioBlock device over MMIO transport.
Type: `VirtioBlockMMIO`.

**FVP_Base_Cortex_A73x1.bp.virtioblockdevice_labeller**
Type: `Labeller`.

**FVP_Base_Cortex_A73x1.bp.virtiop9device**
virtio P9 server.
Type: `VirtioP9Device`.

**FVP_Base_Cortex_A73x1.bp.virtiop9device_labeller**
Type: `Labeller`.

**FVP_Base_Cortex_A73x1.bp.vis**
Display window for VE using Visualisation library.
Type: `VEVisualisation`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A73x1.bp.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_Base_Cortex_A73x1.bp.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x1.bp.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A73x1.bp.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A73x1.bp.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A73x1.bp.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A73x1.bp.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x1.bp.vram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A73x1.cci400**

Cache Coherent Interconnect for AXI4 ACE.
Type: `CCI400`.

**FVP_Base_Cortex_A73x1.clockdivider0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x1.cluster0**

ARM Cortex-A73 Cluster CT model.
Type: `Cluster_ARM_Cortex-A73`.

**FVP_Base_Cortex_A73x1.cluster0.cpu0**

ARM Cortex-A73 CT model.
Type: `ARM_Cortex-A73`.

**FVP_Base_Cortex_A73x1.cluster0.cpu0.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A73x1.cluster0.cpu0.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A73x1.cluster0.cpu0.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A73x1.cluster0.l2_cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A73x1.cluster0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A73x1.dapmemlogger**

Bus Logger.
Type: `PVBusLogger`.

**FVP_Base_Cortex_A73x1.elfloader**

ELF loader component.
Type: `ElfLoader`.

**FVP_Base_Cortex_A73x1.gic_distributor**

GIC Interrupt Redistribution Infrastucture component.
Type: `GIC_IRI`.

**FVP_Base_Cortex_A73x1.pctl**

Base Platforms Power Controller.
Type: `Base_PowerController`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A73x1.pctl.timer_reset**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A73x1.pctl.timer_reset.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A73x1.pctl.timer_reset.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A73x1.pctl.timer_reset.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

## 13.51 FVP_Base_Cortex-A73x1-A53x1

List of instances in FVP_Base_Cortex-A73x1-A53x1.

### FVP_Base_Cortex-A73x1-A53x1 instances

**FVP_Base_Cortex_A73x1_A53x1**

Base Platform Compute Subsystem for ARMCortexA73x1CT and ARMCortexA53x1CT.
Type: `FVP_Base_Cortex_A73x1_A53x1`.

**FVP_Base_Cortex_A73x1_A53x1.bp**

Peripherals and address map for the Base Platform.
Type: `BasePlatformPeripherals`.

**FVP_Base_Cortex_A73x1_A53x1.bp.Timer_0_1**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Cortex_A73x1_A53x1.bp.Timer_0_1.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x1_A53x1.bp.Timer_0_1.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x1_A53x1.bp.Timer_0_1.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A73x1_A53x1.bp.Timer_0_1.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A73x1_A53x1.bp.Timer_2_3**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Cortex_A73x1_A53x1.bp.Timer_2_3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x1_A53x1.bp.Timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x1_A53x1.bp.Timer_2_3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A73x1_A53x1.bp.Timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A73x1_A53x1.bp.ap_refclk**

ARM Generic Timer.
Type: `MemoryMappedGenericTimer`.

**FVP_Base_Cortex_A73x1_A53x1.bp.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A73x1_A53x1.bp.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x1_A53x1.bp.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x1_A53x1.bp.clock300MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x1_A53x1.bp.clock32KHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x1_A53x1.bp.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x1_A53x1.bp.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x1_A53x1.bp.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A73x1_A53x1.bp.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x1_A53x1.bp.dmc**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A73x1_A53x1.bp.dmc_phy**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A73x1_A53x1.bp.dummy_local_dap_rom**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A73x1_A53x1.bp.dummy_ram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A73x1_A53x1.bp.dummy_usb**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A73x1_A53x1.bp.exclusive_monitor**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_Base_Cortex_A73x1_A53x1.bp.flash0**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A73x1_A53x1.bp.flash1**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A73x1_A53x1.bp.flashloader0**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A73x1_A53x1.bp.flashloader1**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A73x1_A53x1.bp.generic_watchdog**

ARM Generic Watchdog.
Type: `MemoryMappedGenericWatchdog`.

**FVP_Base_Cortex_A73x1_A53x1.bp.hdlcd0**

ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `PL370_HDLCD`.

**FVP_Base_Cortex_A73x1_A53x1.bp.hdlcd0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A73x1_A53x1.bp.hdlcd0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A73x1_A53x1.bp.hdlcd0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A73x1_A53x1.bp.hdlcd0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A73x1_A53x1.bp.hdlcd0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A73x1_A53x1.bp.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_Base_Cortex_A73x1_A53x1.bp.mmc**

Generic Multimedia Card.
Type: `MMC`.

**FVP_Base_Cortex_A73x1_A53x1.bp.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A73x1_A53x1.bp.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A73x1_A53x1.bp.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A73x1_A53x1.bp.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A73x1_A53x1.bp.mpe**

Encrypt memory transactions for each encryption context with an independent key to prevent mismatch access.
Type: `PVMemoryProtectionEngine`.

**FVP_Base_Cortex_A73x1_A53x1.bp.nontrustedrom**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A73x1_A53x1.bp.nontrustedromloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A73x1_A53x1.bp.ns_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A73x1_A53x1.bp.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A73x1_A53x1.bp.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x1_A53x1.bp.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A73x1_A53x1.bp.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A73x1_A53x1.bp.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A73x1_A53x1.bp.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A73x1_A53x1.bp.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A73x1_A53x1.bp.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x1_A53x1.bp.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A73x1_A53x1.bp.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A73x1_A53x1.bp.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A73x1_A53x1.bp.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A73x1_A53x1.bp.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A73x1_A53x1.bp.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x1_A53x1.bp.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A73x1_A53x1.bp.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A73x1_A53x1.bp.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A73x1_A53x1.bp.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A73x1_A53x1.bp.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A73x1_A53x1.bp.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x1_A53x1.bp.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A73x1_A53x1.bp.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A73x1_A53x1.bp.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A73x1_A53x1.bp.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A73x1_A53x1.bp.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

**FVP_Base_Cortex_A73x1_A53x1.bp.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A73x1_A53x1.bp.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A73x1_A53x1.bp.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A73x1_A53x1.bp.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A73x1_A53x1.bp.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_Base_Cortex_A73x1_A53x1.bp.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A73x1_A53x1.bp.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A73x1_A53x1.bp.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A73x1_A53x1.bp.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A73x1_A53x1.bp.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A73x1_A53x1.bp.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x1_A53x1.bp.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A73x1_A53x1.bp.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x1_A53x1.bp.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_Base_Cortex_A73x1_A53x1.bp.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

**FVP_Base_Cortex_A73x1_A53x1.bp.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A73x1_A53x1.bp.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A73x1_A53x1.bp.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `SchedulerThread`.

**FVP_Base_Cortex_A73x1_A53x1.bp.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A73x1_A53x1.bp.pl111_clcd_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A73x1_A53x1.bp.pl180_mci**

ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

**FVP_Base_Cortex_A73x1_A53x1.bp.ps2keyboard**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

**FVP_Base_Cortex_A73x1_A53x1.bp.ps2keyboard.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A73x1_A53x1.bp.ps2keyboard.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A73x1_A53x1.bp.ps2keyboard.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A73x1_A53x1.bp.ps2keyboard.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A73x1_A53x1.bp.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `PS2Mouse`.

**FVP_Base_Cortex_A73x1_A53x1.bp.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A73x1_A53x1.bp.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A73x1_A53x1.bp.ps2mouse.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A73x1_A53x1.bp.ps2mouse.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A73x1_A53x1.bp.psram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A73x1_A53x1.bp.refcounter**

Memory Mapped Counter Module for Generic Timers.
Type: `MemoryMappedCounterModule`.

**FVP_Base_Cortex_A73x1_A53x1.bp.reset_or**

Or Gate.
Type: `OrGate`.

**FVP_Base_Cortex_A73x1_A53x1.bp.rl_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A73x1_A53x1.bp.rt_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A73x1_A53x1.bp.s_dram**

RAM device, can be dynamic or static ram.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `RAMDevice`.

**FVP_Base_Cortex_A73x1_A53x1.bp.secureDRAM**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A73x1_A53x1.bp.secureSRAM**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A73x1_A53x1.bp.secureflash**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A73x1_A53x1.bp.secureflashloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A73x1_A53x1.bp.smsc_91c111**

SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_Base_Cortex_A73x1_A53x1.bp.sp805_wdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A73x1_A53x1.bp.sp805_wdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A73x1_A53x1.bp.sp805_wdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A73x1_A53x1.bp.sp805_wdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A73x1_A53x1.bp.sp805_wdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A73x1_A53x1.bp.sp810_sysctrl**

Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_Base_Cortex_A73x1_A53x1.bp.sp810_sysctrl.clkdiv_clk0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x1_A53x1.bp.sp810_sysctrl.clkdiv_clk1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x1_A53x1.bp.sp810_sysctrl.clkdiv_clk2**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x1_A53x1.bp.sp810_sysctrl.clkdiv_clk3**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x1_A53x1.bp.sram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A73x1_A53x1.bp.terminal_0**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A73x1_A53x1.bp.terminal_1**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A73x1_A53x1.bp.terminal_2**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A73x1_A53x1.bp.terminal_3**

Telnet terminal interface.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `TelnetTerminal`.

**FVP_Base_Cortex_A73x1_A53x1.bp.trusted_key_storage**

Trusted Root-Key Storage unit.
Type: `RootKeyStorage`.

**FVP_Base_Cortex_A73x1_A53x1.bp.trusted_nv_counter**

Trusted Non-Volatile Counter unit.
Type: `NonVolatileCounter`.

**FVP_Base_Cortex_A73x1_A53x1.bp.trusted_rng**

Random Number Generator unit.
Type: `RandomNumberGenerator`.

**FVP_Base_Cortex_A73x1_A53x1.bp.trusted_watchdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A73x1_A53x1.bp.trusted_watchdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A73x1_A53x1.bp.trusted_watchdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A73x1_A53x1.bp.trusted_watchdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A73x1_A53x1.bp.trusted_watchdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A73x1_A53x1.bp.tzc_400**

TrustZone Address Space Controller.
Type: `TZC_400`.

**FVP_Base_Cortex_A73x1_A53x1.bp.ve_sysregs**

Type: `VE_SysRegs`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A73x1_A53x1.bp.vedcc**

Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

**FVP_Base_Cortex_A73x1_A53x1.bp.virtio_net**

VirtioNet device over MMIO transport.
Type: `VirtioNetMMIO`.

**FVP_Base_Cortex_A73x1_A53x1.bp.virtio_net_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A73x1_A53x1.bp.virtio_rng**

VirtioEntropy device over MMIO transport.
Type: `VirtioEntropyMMIO`.

**FVP_Base_Cortex_A73x1_A53x1.bp.virtioblockdevice**

VirtioBlock device over MMIO transport.
Type: `VirtioBlockMMIO`.

**FVP_Base_Cortex_A73x1_A53x1.bp.virtioblockdevice_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A73x1_A53x1.bp.virtiop9device**

virtio P9 server.
Type: `VirtioP9Device`.

**FVP_Base_Cortex_A73x1_A53x1.bp.virtiop9device_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A73x1_A53x1.bp.vis**

Display window for VE using Visualisation library.
Type: `VEVisualisation`.

**FVP_Base_Cortex_A73x1_A53x1.bp.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_Base_Cortex_A73x1_A53x1.bp.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x1_A53x1.bp.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ClockTimerThread`.

**FVP_Base_Cortex_A73x1_A53x1.bp.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A73x1_A53x1.bp.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A73x1_A53x1.bp.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A73x1_A53x1.bp.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x1_A53x1.bp.vram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A73x1_A53x1.cci400**

Cache Coherent Interconnect for AXI4 ACE.
Type: `CCI400`.

**FVP_Base_Cortex_A73x1_A53x1.clockdivider0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x1_A53x1.clockdivider1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x1_A53x1.cluster0**

ARM Cortex-A73 Cluster CT model.
Type: `Cluster_ARM_Cortex-A73`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A73x1_A53x1.cluster0.cpu0**
ARM Cortex-A73 CT model.
Type: `ARM_Cortex-A73`.

**FVP_Base_Cortex_A73x1_A53x1.cluster0.cpu0.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A73x1_A53x1.cluster0.cpu0.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A73x1_A53x1.cluster0.cpu0.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A73x1_A53x1.cluster0.l2_cache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A73x1_A53x1.cluster0_labeller**
Type: `Labeller`.

**FVP_Base_Cortex_A73x1_A53x1.cluster1**
ARM Cortex-A53 Cluster CT model.
Type: `Cluster_ARM_Cortex-A53`.

**FVP_Base_Cortex_A73x1_A53x1.cluster1.cpu0**
ARM Cortex-A53 CT model.
Type: `ARM_Cortex-A53`.

**FVP_Base_Cortex_A73x1_A53x1.cluster1.cpu0.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A73x1_A53x1.cluster1.cpu0.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A73x1_A53x1.cluster1.cpu0.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A73x1_A53x1.cluster1.l2_cache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A73x1_A53x1.cluster1_labeller**
Type: `Labeller`.

**FVP_Base_Cortex_A73x1_A53x1.dapmemlogger**
Bus Logger.
Type: `PVBusLogger`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A73x1_A53x1.elfloader**

ELF loader component.
Type: `ElfLoader`.

**FVP_Base_Cortex_A73x1_A53x1.gic_distributor**

GIC Interrupt Redistribution Infrastucture component.
Type: `GIC_IRI`.

**FVP_Base_Cortex_A73x1_A53x1.pctl**

Base Platforms Power Controller.
Type: `Base_PowerController`.

**FVP_Base_Cortex_A73x1_A53x1.pctl.timer_reset**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A73x1_A53x1.pctl.timer_reset.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A73x1_A53x1.pctl.timer_reset.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A73x1_A53x1.pctl.timer_reset.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

## 13.52 FVP_Base_Cortex-A73x2

List of instances in FVP_Base_Cortex-A73x2.

### FVP_Base_Cortex-A73x2 instances

**FVP_Base_Cortex_A73x2**

Base Platform Compute Subsystem for ARMCortexA73x2CT.
Type: `FVP_Base_Cortex_A73x2`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A73x2.bp**

Peripherals and address map for the Base Platform.
Type: `BasePlatformPeripherals`.

**FVP_Base_Cortex_A73x2.bp.Timer_0_1**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Cortex_A73x2.bp.Timer_0_1.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x2.bp.Timer_0_1.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x2.bp.Timer_0_1.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A73x2.bp.Timer_0_1.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A73x2.bp.Timer_2_3**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Cortex_A73x2.bp.Timer_2_3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x2.bp.Timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x2.bp.Timer_2_3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A73x2.bp.Timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A73x2.bp.ap_refclk**

ARM Generic Timer.
Type: `MemoryMappedGenericTimer`.

**FVP_Base_Cortex_A73x2.bp.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**FVP_Base_Cortex_A73x2.bp.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x2.bp.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x2.bp.clock300MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x2.bp.clock32KHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x2.bp.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x2.bp.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x2.bp.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x2.bp.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x2.bp.dmc**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A73x2.bp.dmc_phy**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A73x2.bp.dummy_local_dap_rom**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A73x2.bp.dummy_ram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A73x2.bp.dummy_usb**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A73x2.bp.exclusive_monitor**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_Base_Cortex_A73x2.bp.flash0**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A73x2.bp.flash1**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A73x2.bp.flashloader0**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A73x2.bp.flashloader1**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A73x2.bp.generic_watchdog**

ARM Generic Watchdog.
Type: `MemoryMappedGenericWatchdog`.

**FVP_Base_Cortex_A73x2.bp.hdlcd0**

ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).
Type: `PL370_HDLCD`.

**FVP_Base_Cortex_A73x2.bp.hdlcd0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A73x2.bp.hdlcd0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A73x2.bp.hdlcd0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A73x2.bp.hdlcd0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A73x2.bp.hdlcd0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A73x2.bp.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_Base_Cortex_A73x2.bp.mmc**

Generic Multimedia Card.
Type: `MMC`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A73x2.bp.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A73x2.bp.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A73x2.bp.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A73x2.bp.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A73x2.bp.mpe**

Encrypt memory transactions for each encryption context with an independent key to prevent mismatch access.
Type: `PVMemoryProtectionEngine`.

**FVP_Base_Cortex_A73x2.bp.nontrustedrom**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A73x2.bp.nontrustedromloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A73x2.bp.ns_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A73x2.bp.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A73x2.bp.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x2.bp.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A73x2.bp.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A73x2.bp.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A73x2.bp.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A73x2.bp.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A73x2.bp.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x2.bp.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ClockTimerThread`.

**FVP_Base_Cortex_A73x2.bp.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A73x2.bp.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A73x2.bp.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A73x2.bp.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A73x2.bp.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x2.bp.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A73x2.bp.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A73x2.bp.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A73x2.bp.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A73x2.bp.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A73x2.bp.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x2.bp.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A73x2.bp.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A73x2.bp.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A73x2.bp.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A73x2.bp.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A73x2.bp.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A73x2.bp.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A73x2.bp.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A73x2.bp.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A73x2.bp.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_Base_Cortex_A73x2.bp.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A73x2.bp.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A73x2.bp.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A73x2.bp.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A73x2.bp.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A73x2.bp.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x2.bp.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A73x2.bp.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x2.bp.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_Base_Cortex_A73x2.bp.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

**FVP_Base_Cortex_A73x2.bp.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A73x2.bp.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A73x2.bp.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A73x2.bp.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A73x2.bp.pl111_clcd_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A73x2.bp.pl180_mci**

ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

**FVP_Base_Cortex_A73x2.bp.ps2keyboard**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

**FVP_Base_Cortex_A73x2.bp.ps2keyboard.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A73x2.bp.ps2keyboard.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A73x2.bp.ps2keyboard.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A73x2.bp.ps2keyboard.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A73x2.bp.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

**FVP_Base_Cortex_A73x2.bp.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A73x2.bp.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A73x2.bp.ps2mouse.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A73x2.bp.ps2mouse.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A73x2.bp.psram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A73x2.bp.refcounter**

Memory Mapped Counter Module for Generic Timers.
Type: `MemoryMappedCounterModule`.

**FVP_Base_Cortex_A73x2.bp.reset_or**

Or Gate.
Type: `OrGate`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A73x2.bp.rl_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A73x2.bp.rt_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A73x2.bp.s_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A73x2.bp.secureDRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A73x2.bp.secureSRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A73x2.bp.secureflash**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A73x2.bp.secureflashloader**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A73x2.bp.smsc_91c111**
SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_Base_Cortex_A73x2.bp.sp805_wdog**
ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A73x2.bp.sp805_wdog.clocktimer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A73x2.bp.sp805_wdog.clocktimer.timer**
A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A73x2.bp.sp805_wdog.clocktimer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A73x2.bp.sp805_wdog.clocktimer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A73x2.bp.sp810_sysctrl**
Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_Base_Cortex_A73x2.bp.sp810_sysctrl.clkdiv_clk0**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x2.bp.sp810_sysctrl.clkdiv_clk1**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x2.bp.sp810_sysctrl.clkdiv_clk2**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x2.bp.sp810_sysctrl.clkdiv_clk3**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x2.bp.sram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A73x2.bp.terminal_0**
Telnet terminal interface.
Type: `TelnetTerminal`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A73x2.bp.terminal_1**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A73x2.bp.terminal_2**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A73x2.bp.terminal_3**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A73x2.bp.trusted_key_storage**

Trusted Root-Key Storage unit.
Type: `RootKeyStorage`.

**FVP_Base_Cortex_A73x2.bp.trusted_nv_counter**

Trusted Non-Volatile Counter unit.
Type: `NonVolatileCounter`.

**FVP_Base_Cortex_A73x2.bp.trusted_rng**

Random Number Generator unit.
Type: `RandomNumberGenerator`.

**FVP_Base_Cortex_A73x2.bp.trusted_watchdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A73x2.bp.trusted_watchdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A73x2.bp.trusted_watchdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A73x2.bp.trusted_watchdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A73x2.bp.trusted_watchdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A73x2.bp.tzc_400**

TrustZone Address Space Controller.
Type: `TZC_400`.

**FVP_Base_Cortex_A73x2.bp.ve_sysregs**

Type: `VE_SysRegs`.

**FVP_Base_Cortex_A73x2.bp.vedcc**

Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

**FVP_Base_Cortex_A73x2.bp.virtio_net**

VirtioNet device over MMIO transport.
Type: `VirtioNetMMIO`.

**FVP_Base_Cortex_A73x2.bp.virtio_net_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A73x2.bp.virtio_rng**

VirtioEntropy device over MMIO transport.
Type: `VirtioEntropyMMIO`.

**FVP_Base_Cortex_A73x2.bp.virtioblockdevice**

VirtioBlock device over MMIO transport.
Type: `VirtioBlockMMIO`.

**FVP_Base_Cortex_A73x2.bp.virtioblockdevice_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A73x2.bp.virtiop9device**

virtio P9 server.
Type: `VirtioP9Device`.

**FVP_Base_Cortex_A73x2.bp.virtiop9device_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A73x2.bp.vis**

Display window for VE using Visualisation library.
Type: `VEVisualisation`.

**FVP_Base_Cortex_A73x2.bp.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_Base_Cortex_A73x2.bp.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x2.bp.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A73x2.bp.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A73x2.bp.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A73x2.bp.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A73x2.bp.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x2.bp.vram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A73x2.cci400**

Cache Coherent Interconnect for AXI4 ACE.
Type: `CCI400`.

**FVP_Base_Cortex_A73x2.clockdivider0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x2.cluster0**
ARM Cortex-A73 Cluster CT model.
Type: `Cluster_ARM_Cortex-A73`.

**FVP_Base_Cortex_A73x2.cluster0.cpu0**
ARM Cortex-A73 CT model.
Type: `ARM_Cortex-A73`.

**FVP_Base_Cortex_A73x2.cluster0.cpu0.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A73x2.cluster0.cpu0.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A73x2.cluster0.cpu0.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A73x2.cluster0.cpu1**
ARM Cortex-A73 CT model.
Type: `ARM_Cortex-A73`.

**FVP_Base_Cortex_A73x2.cluster0.cpu1.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A73x2.cluster0.cpu1.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A73x2.cluster0.cpu1.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A73x2.cluster0.l2_cache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A73x2.cluster0_labeller**
Type: `Labeller`.

**FVP_Base_Cortex_A73x2.dapmemlogger**
Bus Logger.
Type: `PVBusLogger`.

**FVP_Base_Cortex_A73x2.elfloader**
ELF loader component.
Type: `ElfLoader`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A73x2.gic_distributor**

GIC Interrupt Redistribution Infrastucture component.
Type: `GIC_IRI`.

**FVP_Base_Cortex_A73x2.pctl**

Base Platforms Power Controller.
Type: `Base_PowerController`.

**FVP_Base_Cortex_A73x2.pctl.timer_reset**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A73x2.pctl.timer_reset.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A73x2.pctl.timer_reset.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A73x2.pctl.timer_reset.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

## 13.53 FVP_Base_Cortex-A73x2-A53x4

List of instances in FVP_Base_Cortex-A73x2-A53x4.

### FVP_Base_Cortex-A73x2-A53x4 instances

**FVP_Base_Cortex_A73x2_A53x4**

Base Platform Compute Subsystem for ARMCortexA73x2CT and ARMCortexA53x4CT.
Type: `FVP_Base_Cortex_A73x2_A53x4`.

**FVP_Base_Cortex_A73x2_A53x4.bp**

Peripherals and address map for the Base Platform.
Type: `BasePlatformPeripherals`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A73x2_A53x4.bp.Timer_0_1**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Cortex_A73x2_A53x4.bp.Timer_0_1.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x2_A53x4.bp.Timer_0_1.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x2_A53x4.bp.Timer_0_1.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A73x2_A53x4.bp.Timer_0_1.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A73x2_A53x4.bp.Timer_2_3**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Cortex_A73x2_A53x4.bp.Timer_2_3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x2_A53x4.bp.Timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x2_A53x4.bp.Timer_2_3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A73x2_A53x4.bp.Timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A73x2_A53x4.bp.ap_refclk**

ARM Generic Timer.
Type: `MemoryMappedGenericTimer`.

**FVP_Base_Cortex_A73x2_A53x4.bp.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**FVP_Base_Cortex_A73x2_A53x4.bp.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x2_A53x4.bp.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x2_A53x4.bp.clock300MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x2_A53x4.bp.clock32KHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x2_A53x4.bp.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x2_A53x4.bp.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A73x2_A53x4.bp.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x2_A53x4.bp.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x2_A53x4.bp.dmc**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A73x2_A53x4.bp.dmc_phy**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A73x2_A53x4.bp.dummy_local_dap_rom**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A73x2_A53x4.bp.dummy_ram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A73x2_A53x4.bp.dummy_usb**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A73x2_A53x4.bp.exclusive_monitor**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_Base_Cortex_A73x2_A53x4.bp.flash0**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A73x2_A53x4.bp.flash1**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A73x2_A53x4.bp.flashloader0**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A73x2_A53x4.bp.flashloader1**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A73x2_A53x4.bp.generic_watchdog**

ARM Generic Watchdog.
Type: `MemoryMappedGenericWatchdog`.

**FVP_Base_Cortex_A73x2_A53x4.bp.hdlcd0**

ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).
Type: `PL370_HDLCD`.

**FVP_Base_Cortex_A73x2_A53x4.bp.hdlcd0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A73x2_A53x4.bp.hdlcd0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A73x2_A53x4.bp.hdlcd0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A73x2_A53x4.bp.hdlcd0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A73x2_A53x4.bp.hdlcd0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A73x2_A53x4.bp.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_Base_Cortex_A73x2_A53x4.bp.mmc**

Generic Multimedia Card.
Type: `MMC`.

**FVP_Base_Cortex_A73x2_A53x4.bp.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A73x2_A53x4.bp.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A73x2_A53x4.bp.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A73x2_A53x4.bp.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A73x2_A53x4.bp.mpe**

Encrypt memory transactions for each encryption context with an independent key to prevent mismatch access.
Type: `PVMemoryProtectionEngine`.

**FVP_Base_Cortex_A73x2_A53x4.bp.nontrustedrom**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A73x2_A53x4.bp.nontrustedromloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A73x2_A53x4.bp.ns_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A73x2_A53x4.bp.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A73x2_A53x4.bp.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A73x2_A53x4.bp.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A73x2_A53x4.bp.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A73x2_A53x4.bp.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A73x2_A53x4.bp.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A73x2_A53x4.bp.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A73x2_A53x4.bp.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x2_A53x4.bp.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A73x2_A53x4.bp.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A73x2_A53x4.bp.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A73x2_A53x4.bp.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A73x2_A53x4.bp.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A73x2_A53x4.bp.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x2_A53x4.bp.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A73x2_A53x4.bp.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A73x2_A53x4.bp.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A73x2_A53x4.bp.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A73x2_A53x4.bp.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A73x2_A53x4.bp.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x2_A53x4.bp.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A73x2_A53x4.bp.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A73x2_A53x4.bp.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A73x2_A53x4.bp.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A73x2_A53x4.bp.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

**FVP_Base_Cortex_A73x2_A53x4.bp.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal()

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A73x2_A53x4.bp.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A73x2_A53x4.bp.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A73x2_A53x4.bp.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A73x2_A53x4.bp.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_Base_Cortex_A73x2_A53x4.bp.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A73x2_A53x4.bp.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A73x2_A53x4.bp.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A73x2_A53x4.bp.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A73x2_A53x4.bp.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A73x2_A53x4.bp.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x2_A53x4.bp.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A73x2_A53x4.bp.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x2_A53x4.bp.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_Base_Cortex_A73x2_A53x4.bp.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

**FVP_Base_Cortex_A73x2_A53x4.bp.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A73x2_A53x4.bp.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A73x2_A53x4.bp.pl111_clcd.pl11x_clcd.timer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A73x2_A53x4.bp.pl111_clcd.pl11x_clcd.timer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A73x2_A53x4.bp.pl111_clcd_labeller**
Type: `Labeller`.

**FVP_Base_Cortex_A73x2_A53x4.bp.pl180_mci**
ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

**FVP_Base_Cortex_A73x2_A53x4.bp.ps2keyboard**
Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

**FVP_Base_Cortex_A73x2_A53x4.bp.ps2keyboard.ps2_clocktimer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A73x2_A53x4.bp.ps2keyboard.ps2_clocktimer.timer**
A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A73x2_A53x4.bp.ps2keyboard.ps2_clocktimer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A73x2_A53x4.bp.ps2keyboard.ps2_clocktimer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A73x2_A53x4.bp.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

**FVP_Base_Cortex_A73x2_A53x4.bp.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A73x2_A53x4.bp.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A73x2_A53x4.bp.ps2mouse.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A73x2_A53x4.bp.ps2mouse.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A73x2_A53x4.bp.psram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A73x2_A53x4.bp.refcounter**

Memory Mapped Counter Module for Generic Timers.
Type: `MemoryMappedCounterModule`.

**FVP_Base_Cortex_A73x2_A53x4.bp.reset_or**

Or Gate.
Type: `OrGate`.

**FVP_Base_Cortex_A73x2_A53x4.bp.rl_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A73x2_A53x4.bp.rt_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A73x2_A53x4.bp.s_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A73x2_A53x4.bp.secureDRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A73x2_A53x4.bp.secureSRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A73x2_A53x4.bp.secureflash**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A73x2_A53x4.bp.secureflashloader**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A73x2_A53x4.bp.smsc_91c111**
SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_Base_Cortex_A73x2_A53x4.bp.sp805_wdog**
ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A73x2_A53x4.bp.sp805_wdog.clocktimer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A73x2_A53x4.bp.sp805_wdog.clocktimer.timer**
A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A73x2_A53x4.bp.sp805_wdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A73x2_A53x4.bp.sp805_wdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A73x2_A53x4.bp.sp810_sysctrl**

Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_Base_Cortex_A73x2_A53x4.bp.sp810_sysctrl.clkdiv_clk0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x2_A53x4.bp.sp810_sysctrl.clkdiv_clk1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x2_A53x4.bp.sp810_sysctrl.clkdiv_clk2**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x2_A53x4.bp.sp810_sysctrl.clkdiv_clk3**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x2_A53x4.bp.sram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A73x2_A53x4.bp.terminal_0**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A73x2_A53x4.bp.terminal_1**

Telnet terminal interface.
Type: `TelnetTerminal`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A73x2_A53x4.bp.terminal_2**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A73x2_A53x4.bp.terminal_3**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A73x2_A53x4.bp.trusted_key_storage**

Trusted Root-Key Storage unit.
Type: `RootKeyStorage`.

**FVP_Base_Cortex_A73x2_A53x4.bp.trusted_nv_counter**

Trusted Non-Volatile Counter unit.
Type: `NonVolatileCounter`.

**FVP_Base_Cortex_A73x2_A53x4.bp.trusted_rng**

Random Number Generator unit.
Type: `RandomNumberGenerator`.

**FVP_Base_Cortex_A73x2_A53x4.bp.trusted_watchdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A73x2_A53x4.bp.trusted_watchdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A73x2_A53x4.bp.trusted_watchdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A73x2_A53x4.bp.trusted_watchdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A73x2_A53x4.bp.trusted_watchdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A73x2_A53x4.bp.tzc_400**

TrustZone Address Space Controller.
Type: `TZC_400`.

**FVP_Base_Cortex_A73x2_A53x4.bp.ve_sysregs**

Type: `VE_SysRegs`.

**FVP_Base_Cortex_A73x2_A53x4.bp.vedcc**

Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

**FVP_Base_Cortex_A73x2_A53x4.bp.virtio_net**

VirtioNet device over MMIO transport.
Type: `VirtioNetMMIO`.

**FVP_Base_Cortex_A73x2_A53x4.bp.virtio_net_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A73x2_A53x4.bp.virtio_rng**

VirtioEntropy device over MMIO transport.
Type: `VirtioEntropyMMIO`.

**FVP_Base_Cortex_A73x2_A53x4.bp.virtioblockdevice**

VirtioBlock device over MMIO transport.
Type: `VirtioBlockMMIO`.

**FVP_Base_Cortex_A73x2_A53x4.bp.virtioblockdevice_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A73x2_A53x4.bp.virtiop9device**

virtio P9 server.
Type: `VirtioP9Device`.

**FVP_Base_Cortex_A73x2_A53x4.bp.virtiop9device_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A73x2_A53x4.bp.vis**

Display window for VE using Visualisation library.
Type: `VEVisualisation`.

**FVP_Base_Cortex_A73x2_A53x4.bp.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_Base_Cortex_A73x2_A53x4.bp.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A73x2_A53x4.bp.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A73x2_A53x4.bp.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A73x2_A53x4.bp.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A73x2_A53x4.bp.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A73x2_A53x4.bp.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x2_A53x4.bp.vram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A73x2_A53x4.cci400**

Cache Coherent Interconnect for AXI4 ACE.
Type: `CCI400`.

**FVP_Base_Cortex_A73x2_A53x4.clockdivider0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A73x2_A53x4.clockdivider1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x2_A53x4.cluster0**

ARM Cortex-A73 Cluster CT model.
Type: `Cluster_ARM_Cortex-A73`.

**FVP_Base_Cortex_A73x2_A53x4.cluster0.cpu0**

ARM Cortex-A73 CT model.
Type: `ARM_Cortex-A73`.

**FVP_Base_Cortex_A73x2_A53x4.cluster0.cpu0.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A73x2_A53x4.cluster0.cpu0.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A73x2_A53x4.cluster0.cpu0.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A73x2_A53x4.cluster0.cpu1**

ARM Cortex-A73 CT model.
Type: `ARM_Cortex-A73`.

**FVP_Base_Cortex_A73x2_A53x4.cluster0.cpu1.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A73x2_A53x4.cluster0.cpu1.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A73x2_A53x4.cluster0.cpu1.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A73x2_A53x4.cluster0.l2_cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A73x2_A53x4.cluster0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A73x2_A53x4.cluster1**

ARM Cortex-A53 Cluster CT model.
Type: `Cluster_ARM_Cortex-A53`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A73x2_A53x4.cluster1.cpu0**

ARM Cortex-A53 CT model.
Type: `ARM_Cortex-A53`.

**FVP_Base_Cortex_A73x2_A53x4.cluster1.cpu0.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A73x2_A53x4.cluster1.cpu0.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A73x2_A53x4.cluster1.cpu0.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A73x2_A53x4.cluster1.cpu1**

ARM Cortex-A53 CT model.
Type: `ARM_Cortex-A53`.

**FVP_Base_Cortex_A73x2_A53x4.cluster1.cpu1.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A73x2_A53x4.cluster1.cpu1.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A73x2_A53x4.cluster1.cpu1.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A73x2_A53x4.cluster1.cpu2**

ARM Cortex-A53 CT model.
Type: `ARM_Cortex-A53`.

**FVP_Base_Cortex_A73x2_A53x4.cluster1.cpu2.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A73x2_A53x4.cluster1.cpu2.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A73x2_A53x4.cluster1.cpu2.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A73x2_A53x4.cluster1.cpu3**

ARM Cortex-A53 CT model.
Type: `ARM_Cortex-A53`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A73x2_A53x4.cluster1.cpu3.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A73x2_A53x4.cluster1.cpu3.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A73x2_A53x4.cluster1.cpu3.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A73x2_A53x4.cluster1.l2_cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A73x2_A53x4.cluster1_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A73x2_A53x4.dapmemlogger**

Bus Logger.
Type: `PVBusLogger`.

**FVP_Base_Cortex_A73x2_A53x4.elfloader**

ELF loader component.
Type: `ElfLoader`.

**FVP_Base_Cortex_A73x2_A53x4.gic_distributor**

GIC Interrupt Redistribution Infrastucture component.
Type: `GIC_IRI`.

**FVP_Base_Cortex_A73x2_A53x4.pctl**

Base Platforms Power Controller.
Type: `Base_PowerController`.

**FVP_Base_Cortex_A73x2_A53x4.pctl.timer_reset**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A73x2_A53x4.pctl.timer_reset.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ClockTimerThread64`.

**`FVP_Base_Cortex_A73x2_A53x4.pctl.timer_reset.timer.thread`**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**`FVP_Base_Cortex_A73x2_A53x4.pctl.timer_reset.timer.thread_event`**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

## 13.54 FVP_Base_Cortex-A73x4

List of instances in FVP_Base_Cortex-A73x4.

### FVP_Base_Cortex-A73x4 instances

**`FVP_Base_Cortex_A73x4`**

Base Platform Compute Subsystem for ARMCortexA73x4CT.
Type: `FVP_Base_Cortex_A73x4`.

**`FVP_Base_Cortex_A73x4.bp`**

Peripherals and address map for the Base Platform.
Type: `BasePlatformPeripherals`.

**`FVP_Base_Cortex_A73x4.bp.Timer_0_1`**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**`FVP_Base_Cortex_A73x4.bp.Timer_0_1.clk_div0`**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**`FVP_Base_Cortex_A73x4.bp.Timer_0_1.clk_div1`**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**`FVP_Base_Cortex_A73x4.bp.Timer_0_1.counter0`**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**`FVP_Base_Cortex_A73x4.bp.Timer_0_1.counter1`**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A73x4.bp.Timer_2_3**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Cortex_A73x4.bp.Timer_2_3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x4.bp.Timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x4.bp.Timer_2_3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A73x4.bp.Timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A73x4.bp.ap_refclk**

ARM Generic Timer.
Type: `MemoryMappedGenericTimer`.

**FVP_Base_Cortex_A73x4.bp.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**FVP_Base_Cortex_A73x4.bp.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x4.bp.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x4.bp.clock300MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x4.bp.clock32KHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x4.bp.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x4.bp.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x4.bp.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x4.bp.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x4.bp.dmc**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A73x4.bp.dmc_phy**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A73x4.bp.dummy_local_dap_rom**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A73x4.bp.dummy_ram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A73x4.bp.dummy_usb**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A73x4.bp.exclusive_monitor**
Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_Base_Cortex_A73x4.bp.flash0**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A73x4.bp.flash1**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A73x4.bp.flashloader0**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A73x4.bp.flashloader1**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A73x4.bp.generic_watchdog**
ARM Generic Watchdog.
Type: `MemoryMappedGenericWatchdog`.

**FVP_Base_Cortex_A73x4.bp.hdlcd0**
ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).
Type: `PL370_HDLCD`.

**FVP_Base_Cortex_A73x4.bp.hdlcd0.timer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A73x4.bp.hdlcd0.timer.timer**
A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A73x4.bp.hdlcd0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A73x4.bp.hdlcd0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A73x4.bp.hdlcd0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A73x4.bp.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_Base_Cortex_A73x4.bp.mmc**

Generic Multimedia Card.
Type: `MMC`.

**FVP_Base_Cortex_A73x4.bp.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A73x4.bp.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A73x4.bp.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A73x4.bp.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A73x4.bp.mpe**

Encrypt memory transactions for each encryption context with an independent key to prevent mismatch access.
Type: `PVMemoryProtectionEngine`.

**FVP_Base_Cortex_A73x4.bp.nontrustedrom**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A73x4.bp.nontrustedromloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A73x4.bp.ns_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A73x4.bp.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A73x4.bp.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x4.bp.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A73x4.bp.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A73x4.bp.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A73x4.bp.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A73x4.bp.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A73x4.bp.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x4.bp.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A73x4.bp.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A73x4.bp.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A73x4.bp.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A73x4.bp.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A73x4.bp.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x4.bp.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A73x4.bp.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A73x4.bp.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A73x4.bp.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A73x4.bp.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A73x4.bp.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x4.bp.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ClockTimerThread`.

**FVP_Base_Cortex_A73x4.bp.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A73x4.bp.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A73x4.bp.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A73x4.bp.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

**FVP_Base_Cortex_A73x4.bp.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A73x4.bp.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A73x4.bp.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A73x4.bp.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A73x4.bp.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_Base_Cortex_A73x4.bp.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A73x4.bp.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A73x4.bp.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A73x4.bp.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A73x4.bp.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A73x4.bp.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x4.bp.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A73x4.bp.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x4.bp.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_Base_Cortex_A73x4.bp.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

**FVP_Base_Cortex_A73x4.bp.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A73x4.bp.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A73x4.bp.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A73x4.bp.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A73x4.bp.pl111_clcd_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A73x4.bp.pl180_mci**

ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

**FVP_Base_Cortex_A73x4.bp.ps2keyboard**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A73x4.bp.ps2keyboard.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A73x4.bp.ps2keyboard.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A73x4.bp.ps2keyboard.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A73x4.bp.ps2keyboard.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A73x4.bp.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

**FVP_Base_Cortex_A73x4.bp.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A73x4.bp.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A73x4.bp.ps2mouse.ps2_clocktimer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A73x4.bp.ps2mouse.ps2_clocktimer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A73x4.bp.psram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A73x4.bp.refcounter**
Memory Mapped Counter Module for Generic Timers.
Type: `MemoryMappedCounterModule`.

**FVP_Base_Cortex_A73x4.bp.reset_or**
Or Gate.
Type: `OrGate`.

**FVP_Base_Cortex_A73x4.bp.rl_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A73x4.bp.rt_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A73x4.bp.s_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A73x4.bp.secureDRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A73x4.bp.secureSRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A73x4.bp.secureflash**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A73x4.bp.secureflashloader**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A73x4.bp.smsc_91c111**

SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_Base_Cortex_A73x4.bp.sp805_wdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A73x4.bp.sp805_wdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A73x4.bp.sp805_wdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A73x4.bp.sp805_wdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A73x4.bp.sp805_wdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A73x4.bp.sp810_sysctrl**

Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_Base_Cortex_A73x4.bp.sp810_sysctrl.clkdiv_clk0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x4.bp.sp810_sysctrl.clkdiv_clk1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x4.bp.sp810_sysctrl.clkdiv_clk2**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x4.bp.sp810_sysctrl.clkdiv_clk3**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x4.bp.sram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A73x4.bp.terminal_0**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A73x4.bp.terminal_1**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A73x4.bp.terminal_2**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A73x4.bp.terminal_3**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A73x4.bp.trusted_key_storage**
Trusted Root-Key Storage unit.
Type: `RootKeyStorage`.

**FVP_Base_Cortex_A73x4.bp.trusted_nv_counter**
Trusted Non-Volatile Counter unit.
Type: `NonVolatileCounter`.

**FVP_Base_Cortex_A73x4.bp.trusted_rng**
Random Number Generator unit.
Type: `RandomNumberGenerator`.

**FVP_Base_Cortex_A73x4.bp.trusted_watchdog**
ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A73x4.bp.trusted_watchdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A73x4.bp.trusted_watchdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A73x4.bp.trusted_watchdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A73x4.bp.trusted_watchdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A73x4.bp.tzc_400**

TrustZone Address Space Controller.
Type: `TZC_400`.

**FVP_Base_Cortex_A73x4.bp.ve_sysregs**

Type: `VE_SysRegs`.

**FVP_Base_Cortex_A73x4.bp.vedcc**

Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

**FVP_Base_Cortex_A73x4.bp.virtio_net**

VirtioNet device over MMIO transport.
Type: `VirtioNetMMIO`.

**FVP_Base_Cortex_A73x4.bp.virtio_net_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A73x4.bp.virtio_rng**

VirtioEntropy device over MMIO transport.
Type: `VirtioEntropyMMIO`.

**FVP_Base_Cortex_A73x4.bp.virtioblockdevice**

VirtioBlock device over MMIO transport.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `VirtioBlockMMIO`.

**FVP_Base_Cortex_A73x4.bp.virtioblockdevice_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A73x4.bp.virtiop9device**

virtio P9 server.
Type: `VirtioP9Device`.

**FVP_Base_Cortex_A73x4.bp.virtiop9device_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A73x4.bp.vis**

Display window for VE using Visualisation library.
Type: `VEVisualisation`.

**FVP_Base_Cortex_A73x4.bp.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_Base_Cortex_A73x4.bp.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x4.bp.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A73x4.bp.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A73x4.bp.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A73x4.bp.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A73x4.bp.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x4.bp.vram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A73x4.cci400**

Cache Coherent Interconnect for AXI4 ACE.
Type: `CCI400`.

**FVP_Base_Cortex_A73x4.clockdivider0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x4.cluster0**

ARM Cortex-A73 Cluster CT model.
Type: `Cluster_ARM_Cortex-A73`.

**FVP_Base_Cortex_A73x4.cluster0.cpu0**

ARM Cortex-A73 CT model.
Type: `ARM_Cortex-A73`.

**FVP_Base_Cortex_A73x4.cluster0.cpu0.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A73x4.cluster0.cpu0.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A73x4.cluster0.cpu0.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A73x4.cluster0.cpu1**

ARM Cortex-A73 CT model.
Type: `ARM_Cortex-A73`.

**FVP_Base_Cortex_A73x4.cluster0.cpu1.dtlb**

TLB - instruction, data or unified.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `TlbCadi`.

**FVP_Base_Cortex_A73x4.cluster0.cpu1.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A73x4.cluster0.cpu1.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A73x4.cluster0.cpu2**
ARM Cortex-A73 CT model.
Type: `ARM_Cortex-A73`.

**FVP_Base_Cortex_A73x4.cluster0.cpu2.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A73x4.cluster0.cpu2.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A73x4.cluster0.cpu2.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A73x4.cluster0.cpu3**
ARM Cortex-A73 CT model.
Type: `ARM_Cortex-A73`.

**FVP_Base_Cortex_A73x4.cluster0.cpu3.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A73x4.cluster0.cpu3.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A73x4.cluster0.cpu3.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A73x4.cluster0.l2_cache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A73x4.cluster0_labeller**
Type: `Labeller`.

**FVP_Base_Cortex_A73x4.dapmemlogger**
Bus Logger.
Type: `PVBusLogger`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A73x4.elfloader**

ELF loader component.
Type: `ElfLoader`.

**FVP_Base_Cortex_A73x4.gic_distributor**

GIC Interrupt Redistribution Infrastucture component.
Type: `GIC_IRI`.

**FVP_Base_Cortex_A73x4.pctl**

Base Platforms Power Controller.
Type: `Base_PowerController`.

**FVP_Base_Cortex_A73x4.pctl.timer_reset**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A73x4.pctl.timer_reset.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A73x4.pctl.timer_reset.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A73x4.pctl.timer_reset.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

## 13.55 FVP_Base_Cortex-A73x4-A53x4

List of instances in FVP_Base_Cortex-A73x4-A53x4.

### FVP_Base_Cortex-A73x4-A53x4 instances

**FVP_Base_Cortex_A73x4_A53x4**

Base Platform Compute Subsystem for ARMCortexA73x4CT and ARMCortexA53x4CT.
Type: `FVP_Base_Cortex_A73x4_A53x4`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A73x4_A53x4.bp**

Peripherals and address map for the Base Platform.
Type: `BasePlatformPeripherals`.

**FVP_Base_Cortex_A73x4_A53x4.bp.Timer_0_1**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Cortex_A73x4_A53x4.bp.Timer_0_1.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x4_A53x4.bp.Timer_0_1.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x4_A53x4.bp.Timer_0_1.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A73x4_A53x4.bp.Timer_0_1.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A73x4_A53x4.bp.Timer_2_3**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Cortex_A73x4_A53x4.bp.Timer_2_3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x4_A53x4.bp.Timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x4_A53x4.bp.Timer_2_3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A73x4_A53x4.bp.Timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A73x4_A53x4.bp.ap_refclk**

ARM Generic Timer.
Type: `MemoryMappedGenericTimer`.

**FVP_Base_Cortex_A73x4_A53x4.bp.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**FVP_Base_Cortex_A73x4_A53x4.bp.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x4_A53x4.bp.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x4_A53x4.bp.clock300MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x4_A53x4.bp.clock32KHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x4_A53x4.bp.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x4_A53x4.bp.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x4_A53x4.bp.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x4_A53x4.bp.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x4_A53x4.bp.dmc**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A73x4_A53x4.bp.dmc_phy**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A73x4_A53x4.bp.dummy_local_dap_rom**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A73x4_A53x4.bp.dummy_ram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A73x4_A53x4.bp.dummy_usb**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A73x4_A53x4.bp.exclusive_monitor**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_Base_Cortex_A73x4_A53x4.bp.flash0**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A73x4_A53x4.bp.flash1**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A73x4_A53x4.bp.flashloader0**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A73x4_A53x4.bp.flashloader1**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A73x4_A53x4.bp.generic_watchdog**

ARM Generic Watchdog.
Type: `MemoryMappedGenericWatchdog`.

**FVP_Base_Cortex_A73x4_A53x4.bp.hdlcd0**

ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).
Type: `PL370_HDLCD`.

**FVP_Base_Cortex_A73x4_A53x4.bp.hdlcd0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A73x4_A53x4.bp.hdlcd0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A73x4_A53x4.bp.hdlcd0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A73x4_A53x4.bp.hdlcd0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A73x4_A53x4.bp.hdlcd0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A73x4_A53x4.bp.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_Base_Cortex_A73x4_A53x4.bp.mmc**

Generic Multimedia Card.
Type: `MMC`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A73x4_A53x4.bp.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A73x4_A53x4.bp.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A73x4_A53x4.bp.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A73x4_A53x4.bp.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A73x4_A53x4.bp.mpe**

Encrypt memory transactions for each encryption context with an independent key to prevent mismatch access.
Type: `PVMemoryProtectionEngine`.

**FVP_Base_Cortex_A73x4_A53x4.bp.nontrustedrom**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A73x4_A53x4.bp.nontrustedromloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A73x4_A53x4.bp.ns_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A73x4_A53x4.bp.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A73x4_A53x4.bp.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x4_A53x4.bp.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A73x4_A53x4.bp.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A73x4_A53x4.bp.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A73x4_A53x4.bp.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A73x4_A53x4.bp.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A73x4_A53x4.bp.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x4_A53x4.bp.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ClockTimerThread`.

**FVP_Base_Cortex_A73x4_A53x4.bp.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A73x4_A53x4.bp.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A73x4_A53x4.bp.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A73x4_A53x4.bp.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A73x4_A53x4.bp.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x4_A53x4.bp.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A73x4_A53x4.bp.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A73x4_A53x4.bp.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A73x4_A53x4.bp.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A73x4_A53x4.bp.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A73x4_A53x4.bp.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x4_A53x4.bp.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A73x4_A53x4.bp.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A73x4_A53x4.bp.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A73x4_A53x4.bp.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A73x4_A53x4.bp.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A73x4_A53x4.bp.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A73x4_A53x4.bp.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A73x4_A53x4.bp.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A73x4_A53x4.bp.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A73x4_A53x4.bp.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_Base_Cortex_A73x4_A53x4.bp.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A73x4_A53x4.bp.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A73x4_A53x4.bp.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A73x4_A53x4.bp.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A73x4_A53x4.bp.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A73x4_A53x4.bp.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x4_A53x4.bp.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A73x4_A53x4.bp.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x4_A53x4.bp.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_Base_Cortex_A73x4_A53x4.bp.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

**FVP_Base_Cortex_A73x4_A53x4.bp.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A73x4_A53x4.bp.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A73x4_A53x4.bp.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A73x4_A53x4.bp.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A73x4_A53x4.bp.pl111_clcd_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A73x4_A53x4.bp.pl180_mci**

ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

**FVP_Base_Cortex_A73x4_A53x4.bp.ps2keyboard**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

**FVP_Base_Cortex_A73x4_A53x4.bp.ps2keyboard.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A73x4_A53x4.bp.ps2keyboard.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A73x4_A53x4.bp.ps2keyboard.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A73x4_A53x4.bp.ps2keyboard.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A73x4_A53x4.bp.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

**FVP_Base_Cortex_A73x4_A53x4.bp.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A73x4_A53x4.bp.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A73x4_A53x4.bp.ps2mouse.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A73x4_A53x4.bp.ps2mouse.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A73x4_A53x4.bp.psram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A73x4_A53x4.bp.refcounter**

Memory Mapped Counter Module for Generic Timers.
Type: `MemoryMappedCounterModule`.

**FVP_Base_Cortex_A73x4_A53x4.bp.reset_or**

Or Gate.
Type: `OrGate`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A73x4_A53x4.bp.rl_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A73x4_A53x4.bp.rt_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A73x4_A53x4.bp.s_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A73x4_A53x4.bp.secureDRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A73x4_A53x4.bp.secureSRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A73x4_A53x4.bp.secureflash**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A73x4_A53x4.bp.secureflashloader**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A73x4_A53x4.bp.smsc_91c111**
SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_Base_Cortex_A73x4_A53x4.bp.sp805_wdog**
ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A73x4_A53x4.bp.sp805_wdog.clocktimer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A73x4_A53x4.bp.sp805_wdog.clocktimer.timer**
A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A73x4_A53x4.bp.sp805_wdog.clocktimer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A73x4_A53x4.bp.sp805_wdog.clocktimer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A73x4_A53x4.bp.sp810_sysctrl**
Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_Base_Cortex_A73x4_A53x4.bp.sp810_sysctrl.clkdiv_clk0**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x4_A53x4.bp.sp810_sysctrl.clkdiv_clk1**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x4_A53x4.bp.sp810_sysctrl.clkdiv_clk2**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x4_A53x4.bp.sp810_sysctrl.clkdiv_clk3**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x4_A53x4.bp.sram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A73x4_A53x4.bp.terminal_0**
Telnet terminal interface.
Type: `TelnetTerminal`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A73x4_A53x4.bp.terminal_1**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A73x4_A53x4.bp.terminal_2**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A73x4_A53x4.bp.terminal_3**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A73x4_A53x4.bp.trusted_key_storage**

Trusted Root-Key Storage unit.
Type: `RootKeyStorage`.

**FVP_Base_Cortex_A73x4_A53x4.bp.trusted_nv_counter**

Trusted Non-Volatile Counter unit.
Type: `NonVolatileCounter`.

**FVP_Base_Cortex_A73x4_A53x4.bp.trusted_rng**

Random Number Generator unit.
Type: `RandomNumberGenerator`.

**FVP_Base_Cortex_A73x4_A53x4.bp.trusted_watchdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A73x4_A53x4.bp.trusted_watchdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A73x4_A53x4.bp.trusted_watchdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A73x4_A53x4.bp.trusted_watchdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A73x4_A53x4.bp.trusted_watchdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A73x4_A53x4.bp.tzc_400**

TrustZone Address Space Controller.
Type: `TZC_400`.

**FVP_Base_Cortex_A73x4_A53x4.bp.ve_sysregs**

Type: `VE_SysRegs`.

**FVP_Base_Cortex_A73x4_A53x4.bp.vedcc**

Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

**FVP_Base_Cortex_A73x4_A53x4.bp.virtio_net**

VirtioNet device over MMIO transport.
Type: `VirtioNetMMIO`.

**FVP_Base_Cortex_A73x4_A53x4.bp.virtio_net_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A73x4_A53x4.bp.virtio_rng**

VirtioEntropy device over MMIO transport.
Type: `VirtioEntropyMMIO`.

**FVP_Base_Cortex_A73x4_A53x4.bp.virtioblockdevice**

VirtioBlock device over MMIO transport.
Type: `VirtioBlockMMIO`.

**FVP_Base_Cortex_A73x4_A53x4.bp.virtioblockdevice_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A73x4_A53x4.bp.virtiop9device**

virtio P9 server.
Type: `VirtioP9Device`.

**FVP_Base_Cortex_A73x4_A53x4.bp.virtiop9device_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A73x4_A53x4.bp.vis**

Display window for VE using Visualisation library.
Type: `VEVisualisation`.

**FVP_Base_Cortex_A73x4_A53x4.bp.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_Base_Cortex_A73x4_A53x4.bp.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x4_A53x4.bp.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A73x4_A53x4.bp.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A73x4_A53x4.bp.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A73x4_A53x4.bp.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A73x4_A53x4.bp.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x4_A53x4.bp.vram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A73x4_A53x4.cci400**

Cache Coherent Interconnect for AXI4 ACE.
Type: `CCI400`.

**FVP_Base_Cortex_A73x4_A53x4.clockdivider0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x4_A53x4.clockdivider1**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A73x4_A53x4.cluster0**
ARM Cortex-A73 Cluster CT model.
Type: `Cluster_ARM_Cortex-A73`.

**FVP_Base_Cortex_A73x4_A53x4.cluster0.cpu0**
ARM Cortex-A73 CT model.
Type: `ARM_Cortex-A73`.

**FVP_Base_Cortex_A73x4_A53x4.cluster0.cpu0.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A73x4_A53x4.cluster0.cpu0.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A73x4_A53x4.cluster0.cpu0.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A73x4_A53x4.cluster0.cpu1**
ARM Cortex-A73 CT model.
Type: `ARM_Cortex-A73`.

**FVP_Base_Cortex_A73x4_A53x4.cluster0.cpu1.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A73x4_A53x4.cluster0.cpu1.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A73x4_A53x4.cluster0.cpu1.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A73x4_A53x4.cluster0.cpu2**
ARM Cortex-A73 CT model.
Type: `ARM_Cortex-A73`.

**FVP_Base_Cortex_A73x4_A53x4.cluster0.cpu2.dtlb**
TLB - instruction, data or unified.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `TlbCadi`.

**FVP_Base_Cortex_A73x4_A53x4.cluster0.cpu2.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A73x4_A53x4.cluster0.cpu2.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A73x4_A53x4.cluster0.cpu3**

ARM Cortex-A73 CT model.
Type: `ARM_Cortex-A73`.

**FVP_Base_Cortex_A73x4_A53x4.cluster0.cpu3.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A73x4_A53x4.cluster0.cpu3.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A73x4_A53x4.cluster0.cpu3.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A73x4_A53x4.cluster0.l2_cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A73x4_A53x4.cluster0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A73x4_A53x4.cluster1**

ARM Cortex-A53 Cluster CT model.
Type: `Cluster_ARM_Cortex-A53`.

**FVP_Base_Cortex_A73x4_A53x4.cluster1.cpu0**

ARM Cortex-A53 CT model.
Type: `ARM_Cortex-A53`.

**FVP_Base_Cortex_A73x4_A53x4.cluster1.cpu0.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A73x4_A53x4.cluster1.cpu0.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A73x4_A53x4.cluster1.cpu0.l1icache**

PV Cache.
Type: `PVCache`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A73x4_A53x4.cluster1.cpu1**

ARM Cortex-A53 CT model.
Type: `ARM_Cortex-A53`.

**FVP_Base_Cortex_A73x4_A53x4.cluster1.cpu1.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A73x4_A53x4.cluster1.cpu1.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A73x4_A53x4.cluster1.cpu1.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A73x4_A53x4.cluster1.cpu2**

ARM Cortex-A53 CT model.
Type: `ARM_Cortex-A53`.

**FVP_Base_Cortex_A73x4_A53x4.cluster1.cpu2.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A73x4_A53x4.cluster1.cpu2.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A73x4_A53x4.cluster1.cpu2.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A73x4_A53x4.cluster1.cpu3**

ARM Cortex-A53 CT model.
Type: `ARM_Cortex-A53`.

**FVP_Base_Cortex_A73x4_A53x4.cluster1.cpu3.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A73x4_A53x4.cluster1.cpu3.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A73x4_A53x4.cluster1.cpu3.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A73x4_A53x4.cluster1.l2_cache**

PV Cache.
Type: `PVCache`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A73x4_A53x4.cluster1_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A73x4_A53x4.dapmemlogger**

Bus Logger.
Type: `PVBusLogger`.

**FVP_Base_Cortex_A73x4_A53x4.elfloader**

ELF loader component.
Type: `ElfLoader`.

**FVP_Base_Cortex_A73x4_A53x4.gic_distributor**

GIC Interrupt Redistribution Infrastucture component.
Type: `GIC_IRI`.

**FVP_Base_Cortex_A73x4_A53x4.pctl**

Base Platforms Power Controller.
Type: `Base_PowerController`.

**FVP_Base_Cortex_A73x4_A53x4.pctl.timer_reset**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A73x4_A53x4.pctl.timer_reset.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A73x4_A53x4.pctl.timer_reset.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A73x4_A53x4.pctl.timer_reset.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

# 13.56 FVP_Base_Cortex-A75

List of instances in FVP_Base_Cortex-A75.

## FVP_Base_Cortex-A75 instances

**FVP_Base_Cortex_A75**

Base Platform Compute Subsystem for ARMCortexA75CT.
Type: `FVP_Base_Cortex_A75`.

**FVP_Base_Cortex_A75.bp**

Peripherals and address map for the Base Platform.
Type: `BasePlatformPeripherals`.

**FVP_Base_Cortex_A75.bp.Timer_0_1**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Cortex_A75.bp.Timer_0_1.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A75.bp.Timer_0_1.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A75.bp.Timer_0_1.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A75.bp.Timer_0_1.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A75.bp.Timer_2_3**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Cortex_A75.bp.Timer_2_3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A75.bp.Timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A75.bp.Timer_2_3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A75.bp.Timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A75.bp.ap_refclk**

ARM Generic Timer.
Type: `MemoryMappedGenericTimer`.

**FVP_Base_Cortex_A75.bp.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**FVP_Base_Cortex_A75.bp.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A75.bp.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A75.bp.clock300MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A75.bp.clock32KHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A75.bp.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A75.bp.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A75.bp.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A75.bp.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A75.bp.dmc**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A75.bp.dmc_phy**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A75.bp.dummy_local_dap_rom**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A75.bp.dummy_ram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A75.bp.dummy_usb**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A75.bp.exclusive_monitor**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A75.bp.flash0**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A75.bp.flash1**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A75.bp.flashloader0**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A75.bp.flashloader1**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A75.bp.generic_watchdog**

ARM Generic Watchdog.
Type: `MemoryMappedGenericWatchdog`.

**FVP_Base_Cortex_A75.bp.hdlcd0**

ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).
Type: `PL370_HDLCD`.

**FVP_Base_Cortex_A75.bp.hdlcd0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A75.bp.hdlcd0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A75.bp.hdlcd0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A75.bp.hdlcd0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A75.bp.hdlcd0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A75.bp.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_Base_Cortex_A75.bp.mmc**

Generic Multimedia Card.
Type: `MMC`.

**FVP_Base_Cortex_A75.bp.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A75.bp.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A75.bp.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A75.bp.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A75.bp.mpe**

Encrypt memory transactions for each encryption context with an independent key to prevent mismatch access.
Type: `PVMemoryProtectionEngine`.

**FVP_Base_Cortex_A75.bp.nontrustedrom**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A75.bp.nontrustedromloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A75.bp.ns_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A75.bp.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A75.bp.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A75.bp.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A75.bp.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A75.bp.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A75.bp.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A75.bp.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A75.bp.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A75.bp.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A75.bp.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A75.bp.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A75.bp.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A75.bp.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A75.bp.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A75.bp.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ClockTimerThread`.

**FVP_Base_Cortex_A75.bp.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A75.bp.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A75.bp.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A75.bp.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A75.bp.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A75.bp.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A75.bp.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A75.bp.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A75.bp.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A75.bp.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

**FVP_Base_Cortex_A75.bp.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A75.bp.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A75.bp.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A75.bp.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A75.bp.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_Base_Cortex_A75.bp.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A75.bp.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A75.bp.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A75.bp.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A75.bp.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A75.bp.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A75.bp.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A75.bp.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A75.bp.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_Base_Cortex_A75.bp.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A75.bp.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A75.bp.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A75.bp.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A75.bp.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A75.bp.pl111_clcd_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A75.bp.pl180_mci**

ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

**FVP_Base_Cortex_A75.bp.ps2keyboard**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

**FVP_Base_Cortex_A75.bp.ps2keyboard.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A75.bp.ps2keyboard.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A75.bp.ps2keyboard.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A75.bp.ps2keyboard.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A75.bp.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

**FVP_Base_Cortex_A75.bp.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A75.bp.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A75.bp.ps2mouse.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A75.bp.ps2mouse.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A75.bp.psram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A75.bp.refcounter**
Memory Mapped Counter Module for Generic Timers.
Type: `MemoryMappedCounterModule`.

**FVP_Base_Cortex_A75.bp.reset_or**
Or Gate.
Type: `OrGate`.

**FVP_Base_Cortex_A75.bp.rl_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A75.bp.rt_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A75.bp.s_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A75.bp.secureDRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A75.bp.secureSRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A75.bp.secureflash**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A75.bp.secureflashloader**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A75.bp.smsc_91c111**
SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_Base_Cortex_A75.bp.sp805_wdog**
ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A75.bp.sp805_wdog.clocktimer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A75.bp.sp805_wdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A75.bp.sp805_wdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A75.bp.sp805_wdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A75.bp.sp810_sysctrl**

Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_Base_Cortex_A75.bp.sp810_sysctrl.clkdiv_clk0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A75.bp.sp810_sysctrl.clkdiv_clk1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A75.bp.sp810_sysctrl.clkdiv_clk2**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A75.bp.sp810_sysctrl.clkdiv_clk3**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A75.bp.sram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A75.bp.terminal_0**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A75.bp.terminal_1**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A75.bp.terminal_2**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A75.bp.terminal_3**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A75.bp.trusted_key_storage**
Trusted Root-Key Storage unit.
Type: `RootKeyStorage`.

**FVP_Base_Cortex_A75.bp.trusted_nv_counter**
Trusted Non-Volatile Counter unit.
Type: `NonVolatileCounter`.

**FVP_Base_Cortex_A75.bp.trusted_rng**
Random Number Generator unit.
Type: `RandomNumberGenerator`.

**FVP_Base_Cortex_A75.bp.trusted_watchdog**
ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A75.bp.trusted_watchdog.clocktimer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A75.bp.trusted_watchdog.clocktimer.timer**
A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A75.bp.trusted_watchdog.clocktimer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A75.bp.trusted_watchdog.clocktimer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A75.bp.tzc_400**
TrustZone Address Space Controller.
Type: `TZC_400`.

**FVP_Base_Cortex_A75.bp.ve_sysregs**
Type: `VE_SysRegs`.

**FVP_Base_Cortex_A75.bp.vedcc**
Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

**FVP_Base_Cortex_A75.bp.virtio_net**
VirtioNet device over MMIO transport.
Type: `VirtioNetMMIO`.

**FVP_Base_Cortex_A75.bp.virtio_net_labeller**
Type: `Labeller`.

**FVP_Base_Cortex_A75.bp.virtio_rng**
VirtioEntropy device over MMIO transport.
Type: `VirtioEntropyMMIO`.

**FVP_Base_Cortex_A75.bp.virtioblockdevice**
VirtioBlock device over MMIO transport.
Type: `VirtioBlockMMIO`.

**FVP_Base_Cortex_A75.bp.virtioblockdevice_labeller**
Type: `Labeller`.

**FVP_Base_Cortex_A75.bp.virtiop9device**
virtio P9 server.
Type: `VirtioP9Device`.

**FVP_Base_Cortex_A75.bp.virtiop9device_labeller**
Type: `Labeller`.

**FVP_Base_Cortex_A75.bp.vis**
Display window for VE using Visualisation library.
Type: `VEVisualisation`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A75.bp.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_Base_Cortex_A75.bp.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A75.bp.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A75.bp.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A75.bp.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A75.bp.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A75.bp.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A75.bp.vram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A75.cci400**

Cache Coherent Interconnect for AXI4 ACE.
Type: `CCI400`.

**FVP_Base_Cortex_A75.clockdivider0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A75.cluster0**

ARM Cortex-A75 Cluster CT model.
Type: `Cluster_ARM_Cortex-A75`.

**FVP_Base_Cortex_A75.cluster0.DSU.shared_cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A75.cluster0.cpu0**

ARM Cortex-A75 CT model.
Type: `ARM_Cortex-A75`.

**FVP_Base_Cortex_A75.cluster0.cpu0.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A75.cluster0.cpu0.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A75.cluster0.cpu0.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A75.cluster0.cpu0.l2cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A75.cluster0.cpu1**

ARM Cortex-A75 CT model.
Type: `ARM_Cortex-A75`.

**FVP_Base_Cortex_A75.cluster0.cpu1.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A75.cluster0.cpu1.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A75.cluster0.cpu1.l1icache**

PV Cache.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `PVCache`.

**FVP_Base_Cortex_A75.cluster0.cpu1.l2cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A75.cluster0.cpu2**

ARM Cortex-A75 CT model.
Type: `ARM_Cortex-A75`.

**FVP_Base_Cortex_A75.cluster0.cpu2.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A75.cluster0.cpu2.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A75.cluster0.cpu2.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A75.cluster0.cpu2.l2cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A75.cluster0.cpu3**

ARM Cortex-A75 CT model.
Type: `ARM_Cortex-A75`.

**FVP_Base_Cortex_A75.cluster0.cpu3.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A75.cluster0.cpu3.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A75.cluster0.cpu3.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A75.cluster0.cpu3.l2cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A75.cluster0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A75.dapmemlogger**

Bus Logger.
Type: `PVBusLogger`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A75.elfloader**

ELF loader component.
Type: `ElfLoader`.

**FVP_Base_Cortex_A75.gic_distributor**

GIC Interrupt Redistribution Infrastucture component.
Type: `GIC_IRI`.

**FVP_Base_Cortex_A75.pctl**

Base Platforms Power Controller.
Type: `Base_PowerController`.

**FVP_Base_Cortex_A75.pctl.timer_reset**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A75.pctl.timer_reset.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A75.pctl.timer_reset.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A75.pctl.timer_reset.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

# 13.57 FVP_Base_Cortex-A76

List of instances in FVP_Base_Cortex-A76.

## FVP_Base_Cortex-A76 instances

**FVP_Base_Cortex_A76**

Base Platform Compute Subsystem for ARMCortexA76CT.
Type: `FVP_Base_Cortex_A76`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A76.bp**

Peripherals and address map for the Base Platform.
Type: `BasePlatformPeripherals`.

**FVP_Base_Cortex_A76.bp.Timer_0_1**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Cortex_A76.bp.Timer_0_1.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A76.bp.Timer_0_1.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A76.bp.Timer_0_1.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A76.bp.Timer_0_1.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A76.bp.Timer_2_3**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Cortex_A76.bp.Timer_2_3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A76.bp.Timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A76.bp.Timer_2_3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A76.bp.Timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A76.bp.ap_refclk**

ARM Generic Timer.
Type: `MemoryMappedGenericTimer`.

**FVP_Base_Cortex_A76.bp.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**FVP_Base_Cortex_A76.bp.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A76.bp.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A76.bp.clock300MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A76.bp.clock32KHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A76.bp.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A76.bp.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A76.bp.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A76.bp.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A76.bp.dmc**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A76.bp.dmc_phy**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A76.bp.dummy_local_dap_rom**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A76.bp.dummy_ram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A76.bp.dummy_usb**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A76.bp.exclusive_monitor**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_Base_Cortex_A76.bp.flash0**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A76.bp.flash1**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A76.bp.flashloader0**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A76.bp.flashloader1**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A76.bp.generic_watchdog**

ARM Generic Watchdog.
Type: `MemoryMappedGenericWatchdog`.

**FVP_Base_Cortex_A76.bp.hdlcd0**

ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).
Type: `PL370_HDLCD`.

**FVP_Base_Cortex_A76.bp.hdlcd0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A76.bp.hdlcd0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A76.bp.hdlcd0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A76.bp.hdlcd0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A76.bp.hdlcd0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A76.bp.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_Base_Cortex_A76.bp.mmc**

Generic Multimedia Card.
Type: `MMC`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A76.bp.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A76.bp.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A76.bp.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A76.bp.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A76.bp.mpe**

Encrypt memory transactions for each encryption context with an independent key to prevent mismatch access.
Type: `PVMemoryProtectionEngine`.

**FVP_Base_Cortex_A76.bp.nontrustedrom**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A76.bp.nontrustedromloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A76.bp.ns_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A76.bp.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A76.bp.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A76.bp.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A76.bp.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A76.bp.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A76.bp.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A76.bp.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A76.bp.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A76.bp.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ClockTimerThread`.

**FVP_Base_Cortex_A76.bp.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A76.bp.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A76.bp.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A76.bp.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A76.bp.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A76.bp.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A76.bp.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A76.bp.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A76.bp.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A76.bp.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A76.bp.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A76.bp.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A76.bp.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A76.bp.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A76.bp.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A76.bp.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A76.bp.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A76.bp.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A76.bp.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A76.bp.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A76.bp.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_Base_Cortex_A76.bp.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A76.bp.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A76.bp.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A76.bp.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A76.bp.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A76.bp.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A76.bp.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A76.bp.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A76.bp.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_Base_Cortex_A76.bp.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

**FVP_Base_Cortex_A76.bp.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A76.bp.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A76.bp.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A76.bp.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A76.bp.pl111_clcd_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A76.bp.pl180_mci**

ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

**FVP_Base_Cortex_A76.bp.ps2keyboard**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

**FVP_Base_Cortex_A76.bp.ps2keyboard.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A76.bp.ps2keyboard.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A76.bp.ps2keyboard.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A76.bp.ps2keyboard.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A76.bp.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

**FVP_Base_Cortex_A76.bp.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A76.bp.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A76.bp.ps2mouse.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A76.bp.ps2mouse.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A76.bp.psram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A76.bp.refcounter**

Memory Mapped Counter Module for Generic Timers.
Type: `MemoryMappedCounterModule`.

**FVP_Base_Cortex_A76.bp.reset_or**

Or Gate.
Type: `OrGate`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A76.bp.rl_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A76.bp.rt_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A76.bp.s_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A76.bp.secureDRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A76.bp.secureSRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A76.bp.secureflash**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A76.bp.secureflashloader**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A76.bp.smsc_91c111**
SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_Base_Cortex_A76.bp.sp805_wdog**
ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A76.bp.sp805_wdog.clocktimer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A76.bp.sp805_wdog.clocktimer.timer**
A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A76.bp.sp805_wdog.clocktimer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A76.bp.sp805_wdog.clocktimer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A76.bp.sp810_sysctrl**
Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_Base_Cortex_A76.bp.sp810_sysctrl.clkdiv_clk0**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A76.bp.sp810_sysctrl.clkdiv_clk1**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A76.bp.sp810_sysctrl.clkdiv_clk2**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A76.bp.sp810_sysctrl.clkdiv_clk3**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A76.bp.sram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A76.bp.terminal_0**
Telnet terminal interface.
Type: `TelnetTerminal`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A76.bp.terminal_1**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A76.bp.terminal_2**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A76.bp.terminal_3**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A76.bp.trusted_key_storage**

Trusted Root-Key Storage unit.
Type: `RootKeyStorage`.

**FVP_Base_Cortex_A76.bp.trusted_nv_counter**

Trusted Non-Volatile Counter unit.
Type: `NonVolatileCounter`.

**FVP_Base_Cortex_A76.bp.trusted_rng**

Random Number Generator unit.
Type: `RandomNumberGenerator`.

**FVP_Base_Cortex_A76.bp.trusted_watchdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A76.bp.trusted_watchdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A76.bp.trusted_watchdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A76.bp.trusted_watchdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A76.bp.trusted_watchdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A76.bp.tzc_400**

TrustZone Address Space Controller.
Type: `TZC_400`.

**FVP_Base_Cortex_A76.bp.ve_sysregs**

Type: `VE_SysRegs`.

**FVP_Base_Cortex_A76.bp.vedcc**

Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

**FVP_Base_Cortex_A76.bp.virtio_net**

VirtioNet device over MMIO transport.
Type: `VirtioNetMMIO`.

**FVP_Base_Cortex_A76.bp.virtio_net_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A76.bp.virtio_rng**

VirtioEntropy device over MMIO transport.
Type: `VirtioEntropyMMIO`.

**FVP_Base_Cortex_A76.bp.virtioblockdevice**

VirtioBlock device over MMIO transport.
Type: `VirtioBlockMMIO`.

**FVP_Base_Cortex_A76.bp.virtioblockdevice_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A76.bp.virtiop9device**

virtio P9 server.
Type: `VirtioP9Device`.

**FVP_Base_Cortex_A76.bp.virtiop9device_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A76.bp.vis**

Display window for VE using Visualisation library.
Type: `VEVisualisation`.

**FVP_Base_Cortex_A76.bp.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_Base_Cortex_A76.bp.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A76.bp.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A76.bp.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A76.bp.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A76.bp.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A76.bp.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A76.bp.vram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A76.cci400**

Cache Coherent Interconnect for AXI4 ACE.
Type: `CCI400`.

**FVP_Base_Cortex_A76.clockdivider0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A76.cluster0**
ARM Cortex-A76 Cluster CT model.
Type: `Cluster_ARM_Cortex-A76`.

**FVP_Base_Cortex_A76.cluster0.DSU.shared_cache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A76.cluster0.cpu0**
ARM Cortex-A76 CT model.
Type: `ARM_Cortex-A76`.

**FVP_Base_Cortex_A76.cluster0.cpu0.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A76.cluster0.cpu0.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A76.cluster0.cpu0.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A76.cluster0.cpu0.l2cache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A76.cluster0.cpu1**
ARM Cortex-A76 CT model.
Type: `ARM_Cortex-A76`.

**FVP_Base_Cortex_A76.cluster0.cpu1.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A76.cluster0.cpu1.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A76.cluster0.cpu1.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A76.cluster0.cpu1.l2cache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A76.cluster0.cpu2**
ARM Cortex-A76 CT model.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ARM_Cortex-A76`.

**FVP_Base_Cortex_A76.cluster0.cpu2.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A76.cluster0.cpu2.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A76.cluster0.cpu2.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A76.cluster0.cpu2.l2cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A76.cluster0.cpu3**

ARM Cortex-A76 CT model.
Type: `ARM_Cortex-A76`.

**FVP_Base_Cortex_A76.cluster0.cpu3.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A76.cluster0.cpu3.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A76.cluster0.cpu3.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A76.cluster0.cpu3.l2cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A76.cluster0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A76.dapmemlogger**

Bus Logger.
Type: `PVBusLogger`.

**FVP_Base_Cortex_A76.elfloader**

ELF loader component.
Type: `ElfLoader`.

**FVP_Base_Cortex_A76.gic_distributor**

GIC Interrupt Redistribution Infrastucture component.
Type: `GIC_IRI`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A76.pctl**

Base Platforms Power Controller.
Type: `Base_PowerController`.

**FVP_Base_Cortex_A76.pctl.timer_reset**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A76.pctl.timer_reset.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A76.pctl.timer_reset.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A76.pctl.timer_reset.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

## 13.58 FVP_Base_Cortex-A76AE

List of instances in FVP_Base_Cortex-A76AE.

### FVP_Base_Cortex-A76AE instances

**FVP_Base_Cortex_A76AE**

Base Platform Compute Subsystem for ARMCortexA76AECT.
Type: `FVP_Base_Cortex_A76AE`.

**FVP_Base_Cortex_A76AE.bp**

Peripherals and address map for the Base Platform.
Type: `BasePlatformPeripherals`.

**FVP_Base_Cortex_A76AE.bp.Timer_0_1**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A76AE.bp.Timer_0_1.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A76AE.bp.Timer_0_1.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A76AE.bp.Timer_0_1.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A76AE.bp.Timer_0_1.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A76AE.bp.Timer_2_3**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Cortex_A76AE.bp.Timer_2_3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A76AE.bp.Timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A76AE.bp.Timer_2_3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A76AE.bp.Timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A76AE.bp.ap_refclk**

ARM Generic Timer.
Type: `MemoryMappedGenericTimer`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A76AE.bp.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**FVP_Base_Cortex_A76AE.bp.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A76AE.bp.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A76AE.bp.clock300MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A76AE.bp.clock32KHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A76AE.bp.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A76AE.bp.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A76AE.bp.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ClockDivider`.

**FVP_Base_Cortex_A76AE.bp.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A76AE.bp.dmc**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A76AE.bp.dmc_phy**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A76AE.bp.dummy_local_dap_rom**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A76AE.bp.dummy_ram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A76AE.bp.dummy_usb**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A76AE.bp.exclusive_monitor**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_Base_Cortex_A76AE.bp.flash0**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A76AE.bp.flash1**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A76AE.bp.flashloader0**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A76AE.bp.flashloader1**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A76AE.bp.generic_watchdog**

ARM Generic Watchdog.
Type: `MemoryMappedGenericWatchdog`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A76AE.bp.hdlcd0**

ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).
Type: `PL370_HDLCD`.

**FVP_Base_Cortex_A76AE.bp.hdlcd0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A76AE.bp.hdlcd0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A76AE.bp.hdlcd0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A76AE.bp.hdlcd0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A76AE.bp.hdlcd0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A76AE.bp.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_Base_Cortex_A76AE.bp.mmc**

Generic Multimedia Card.
Type: `MMC`.

**FVP_Base_Cortex_A76AE.bp.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ClockTimerThread`.

**FVP_Base_Cortex_A76AE.bp.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A76AE.bp.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A76AE.bp.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A76AE.bp.mpe**

Encrypt memory transactions for each encryption context with an independent key to prevent mismatch access.
Type: `PVMemoryProtectionEngine`.

**FVP_Base_Cortex_A76AE.bp.nontrustedrom**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A76AE.bp.nontrustedromloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A76AE.bp.ns_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A76AE.bp.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A76AE.bp.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A76AE.bp.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A76AE.bp.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A76AE.bp.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A76AE.bp.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A76AE.bp.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A76AE.bp.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A76AE.bp.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A76AE.bp.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A76AE.bp.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A76AE.bp.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A76AE.bp.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A76AE.bp.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A76AE.bp.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A76AE.bp.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A76AE.bp.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A76AE.bp.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A76AE.bp.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A76AE.bp.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A76AE.bp.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A76AE.bp.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A76AE.bp.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A76AE.bp.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A76AE.bp.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

**FVP_Base_Cortex_A76AE.bp.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ClockTimerThread`.

**FVP_Base_Cortex_A76AE.bp.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A76AE.bp.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A76AE.bp.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A76AE.bp.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_Base_Cortex_A76AE.bp.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A76AE.bp.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A76AE.bp.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A76AE.bp.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A76AE.bp.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A76AE.bp.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A76AE.bp.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A76AE.bp.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A76AE.bp.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_Base_Cortex_A76AE.bp.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

**FVP_Base_Cortex_A76AE.bp.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A76AE.bp.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A76AE.bp.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `SchedulerThread`.

**FVP_Base_Cortex_A76AE.bp.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A76AE.bp.pl111_clcd_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A76AE.bp.pl180_mci**

ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

**FVP_Base_Cortex_A76AE.bp.ps2keyboard**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

**FVP_Base_Cortex_A76AE.bp.ps2keyboard.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A76AE.bp.ps2keyboard.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A76AE.bp.ps2keyboard.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A76AE.bp.ps2keyboard.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A76AE.bp.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `PS2Mouse`.

**FVP_Base_Cortex_A76AE.bp.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A76AE.bp.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A76AE.bp.ps2mouse.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A76AE.bp.ps2mouse.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A76AE.bp.psram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A76AE.bp.refcounter**

Memory Mapped Counter Module for Generic Timers.
Type: `MemoryMappedCounterModule`.

**FVP_Base_Cortex_A76AE.bp.reset_or**

Or Gate.
Type: `OrGate`.

**FVP_Base_Cortex_A76AE.bp.rl_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A76AE.bp.rt_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A76AE.bp.s_dram**

RAM device, can be dynamic or static ram.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `RAMDevice`.

**FVP_Base_Cortex_A76AE.bp.secureDRAM**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A76AE.bp.secureSRAM**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A76AE.bp.secureflash**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A76AE.bp.secureflashloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A76AE.bp.smsc_91c111**

SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_Base_Cortex_A76AE.bp.sp805_wdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A76AE.bp.sp805_wdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A76AE.bp.sp805_wdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A76AE.bp.sp805_wdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A76AE.bp.sp805_wdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A76AE.bp.sp810_sysctrl**

Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_Base_Cortex_A76AE.bp.sp810_sysctrl.clkdiv_clk0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A76AE.bp.sp810_sysctrl.clkdiv_clk1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A76AE.bp.sp810_sysctrl.clkdiv_clk2**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A76AE.bp.sp810_sysctrl.clkdiv_clk3**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A76AE.bp.sram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A76AE.bp.terminal_0**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A76AE.bp.terminal_1**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A76AE.bp.terminal_2**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A76AE.bp.terminal_3**

Telnet terminal interface.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `TelnetTerminal`.

**FVP_Base_Cortex_A76AE.bp.trusted_key_storage**

Trusted Root-Key Storage unit.
Type: `RootKeyStorage`.

**FVP_Base_Cortex_A76AE.bp.trusted_nv_counter**

Trusted Non-Volatile Counter unit.
Type: `NonVolatileCounter`.

**FVP_Base_Cortex_A76AE.bp.trusted_rng**

Random Number Generator unit.
Type: `RandomNumberGenerator`.

**FVP_Base_Cortex_A76AE.bp.trusted_watchdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A76AE.bp.trusted_watchdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A76AE.bp.trusted_watchdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A76AE.bp.trusted_watchdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A76AE.bp.trusted_watchdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A76AE.bp.tzc_400**

TrustZone Address Space Controller.
Type: `TZC_400`.

**FVP_Base_Cortex_A76AE.bp.ve_sysregs**

Type: `VE_SysRegs`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A76AE.bp.vedcc**

Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

**FVP_Base_Cortex_A76AE.bp.virtio_net**

VirtioNet device over MMIO transport.
Type: `VirtioNetMMIO`.

**FVP_Base_Cortex_A76AE.bp.virtio_net_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A76AE.bp.virtio_rng**

VirtioEntropy device over MMIO transport.
Type: `VirtioEntropyMMIO`.

**FVP_Base_Cortex_A76AE.bp.virtioblockdevice**

VirtioBlock device over MMIO transport.
Type: `VirtioBlockMMIO`.

**FVP_Base_Cortex_A76AE.bp.virtioblockdevice_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A76AE.bp.virtiop9device**

virtio P9 server.
Type: `VirtioP9Device`.

**FVP_Base_Cortex_A76AE.bp.virtiop9device_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A76AE.bp.vis**

Display window for VE using Visualisation library.
Type: `VEVisualisation`.

**FVP_Base_Cortex_A76AE.bp.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_Base_Cortex_A76AE.bp.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A76AE.bp.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ClockTimerThread`.

**FVP_Base_Cortex_A76AE.bp.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A76AE.bp.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A76AE.bp.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A76AE.bp.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A76AE.bp.vram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A76AE.cci400**

Cache Coherent Interconnect for AXI4 ACE.
Type: `CCI400`.

**FVP_Base_Cortex_A76AE.clockdivider0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A76AE.cluster0**

ARM Cortex-A76AE Cluster CT model.
Type: `Cluster_ARM_Cortex-A76AE`.

**FVP_Base_Cortex_A76AE.cluster0.DSU.shared_cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A76AE.cluster0.cpu0**

ARM Cortex-A76AE CT model.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ARM_Cortex-A76AE`.

**FVP_Base_Cortex_A76AE.cluster0.cpu0.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A76AE.cluster0.cpu0.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A76AE.cluster0.cpu0.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A76AE.cluster0.cpu0.l2cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A76AE.cluster0.cpu1**

ARM Cortex-A76AE CT model.
Type: `ARM_Cortex-A76AE`.

**FVP_Base_Cortex_A76AE.cluster0.cpu1.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A76AE.cluster0.cpu1.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A76AE.cluster0.cpu1.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A76AE.cluster0.cpu1.l2cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A76AE.cluster0.cpu2**

ARM Cortex-A76AE CT model.
Type: `ARM_Cortex-A76AE`.

**FVP_Base_Cortex_A76AE.cluster0.cpu2.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A76AE.cluster0.cpu2.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A76AE.cluster0.cpu2.l1icache**

PV Cache.
Type: `PVCache`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A76AE.cluster0.cpu2.l2cache**

PV Cache.
Type: PVCache.

**FVP_Base_Cortex_A76AE.cluster0.cpu3**

ARM Cortex-A76AE CT model.
Type: ARM_Cortex-A76AE.

**FVP_Base_Cortex_A76AE.cluster0.cpu3.dtlb**

TLB - instruction, data or unified.
Type: TlbCadi.

**FVP_Base_Cortex_A76AE.cluster0.cpu3.l1dcache**

PV Cache.
Type: PVCache.

**FVP_Base_Cortex_A76AE.cluster0.cpu3.l1icache**

PV Cache.
Type: PVCache.

**FVP_Base_Cortex_A76AE.cluster0.cpu3.l2cache**

PV Cache.
Type: PVCache.

**FVP_Base_Cortex_A76AE.cluster0_labeller**

Type: Labeller.

**FVP_Base_Cortex_A76AE.dapmemlogger**

Bus Logger.
Type: PVBusLogger.

**FVP_Base_Cortex_A76AE.elfloader**

ELF loader component.
Type: ElfLoader.

**FVP_Base_Cortex_A76AE.gic_distributor**

GIC Interrupt Redistribution Infrastucture component.
Type: GIC_IRI.

**FVP_Base_Cortex_A76AE.pctl**

Base Platforms Power Controller.
Type: Base_PowerController.

**FVP_Base_Cortex_A76AE.pctl.timer_reset**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: ClockTimerThread.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A76AE.pctl.timer_reset.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A76AE.pctl.timer_reset.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A76AE.pctl.timer_reset.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

## 13.59 FVP_Base_Cortex-A77

List of instances in FVP_Base_Cortex-A77.

### FVP_Base_Cortex-A77 instances

**FVP_Base_Cortex_A77**

Base Platform Compute Subsystem for ARMCortexA77CT.
Type: `FVP_Base_Cortex_A77`.

**FVP_Base_Cortex_A77.bp**

Peripherals and address map for the Base Platform.
Type: `BasePlatformPeripherals`.

**FVP_Base_Cortex_A77.bp.Timer_0_1**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Cortex_A77.bp.Timer_0_1.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A77.bp.Timer_0_1.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A77.bp.Timer_0_1.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A77.bp.Timer_0_1.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A77.bp.Timer_2_3**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Cortex_A77.bp.Timer_2_3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A77.bp.Timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A77.bp.Timer_2_3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A77.bp.Timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A77.bp.ap_refclk**

ARM Generic Timer.
Type: `MemoryMappedGenericTimer`.

**FVP_Base_Cortex_A77.bp.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**FVP_Base_Cortex_A77.bp.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A77.bp.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A77.bp.clock300MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A77.bp.clock32KHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A77.bp.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A77.bp.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A77.bp.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A77.bp.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A77.bp.dmc**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A77.bp.dmc_phy**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A77.bp.dummy_local_dap_rom**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A77.bp.dummy_ram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A77.bp.dummy_usb**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A77.bp.exclusive_monitor**
Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_Base_Cortex_A77.bp.flash0**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A77.bp.flash1**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A77.bp.flashloader0**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A77.bp.flashloader1**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A77.bp.generic_watchdog**
ARM Generic Watchdog.
Type: `MemoryMappedGenericWatchdog`.

**FVP_Base_Cortex_A77.bp.hdlcd0**
ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).
Type: `PL370_HDLCD`.

**FVP_Base_Cortex_A77.bp.hdlcd0.timer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A77.bp.hdlcd0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A77.bp.hdlcd0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A77.bp.hdlcd0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A77.bp.hdlcd0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A77.bp.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_Base_Cortex_A77.bp.mmc**

Generic Multimedia Card.
Type: `MMC`.

**FVP_Base_Cortex_A77.bp.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A77.bp.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A77.bp.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A77.bp.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A77.bp.mpe**

Encrypt memory transactions for each encryption context with an independent key to prevent mismatch access.
Type: `PVMemoryProtectionEngine`.

**FVP_Base_Cortex_A77.bp.nontrustedrom**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A77.bp.nontrustedromloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A77.bp.ns_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A77.bp.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A77.bp.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A77.bp.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A77.bp.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A77.bp.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A77.bp.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A77.bp.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A77.bp.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A77.bp.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A77.bp.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A77.bp.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A77.bp.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A77.bp.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A77.bp.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A77.bp.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A77.bp.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A77.bp.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A77.bp.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A77.bp.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A77.bp.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A77.bp.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A77.bp.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A77.bp.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A77.bp.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A77.bp.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

**FVP_Base_Cortex_A77.bp.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A77.bp.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A77.bp.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A77.bp.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A77.bp.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_Base_Cortex_A77.bp.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A77.bp.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A77.bp.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A77.bp.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A77.bp.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A77.bp.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A77.bp.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A77.bp.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A77.bp.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_Base_Cortex_A77.bp.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

**FVP_Base_Cortex_A77.bp.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A77.bp.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A77.bp.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A77.bp.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A77.bp.pl111_clcd_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A77.bp.pl180_mci**

ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A77.bp.ps2keyboard**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

**FVP_Base_Cortex_A77.bp.ps2keyboard.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A77.bp.ps2keyboard.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A77.bp.ps2keyboard.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A77.bp.ps2keyboard.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A77.bp.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

**FVP_Base_Cortex_A77.bp.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A77.bp.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A77.bp.ps2mouse.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A77.bp.ps2mouse.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A77.bp.psram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A77.bp.refcounter**

Memory Mapped Counter Module for Generic Timers.
Type: `MemoryMappedCounterModule`.

**FVP_Base_Cortex_A77.bp.reset_or**

Or Gate.
Type: `OrGate`.

**FVP_Base_Cortex_A77.bp.rl_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A77.bp.rt_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A77.bp.s_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A77.bp.secureDRAM**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A77.bp.secureSRAM**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A77.bp.secureflash**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A77.bp.secureflashloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A77.bp.smsc_91c111**

SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_Base_Cortex_A77.bp.sp805_wdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A77.bp.sp805_wdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A77.bp.sp805_wdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A77.bp.sp805_wdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A77.bp.sp805_wdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A77.bp.sp810_sysctrl**

Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_Base_Cortex_A77.bp.sp810_sysctrl.clkdiv_clk0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A77.bp.sp810_sysctrl.clkdiv_clk1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A77.bp.sp810_sysctrl.clkdiv_clk2**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A77.bp.sp810_sysctrl.clkdiv_clk3**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A77.bp.sram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A77.bp.terminal_0**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A77.bp.terminal_1**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A77.bp.terminal_2**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A77.bp.terminal_3**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A77.bp.trusted_key_storage**

Trusted Root-Key Storage unit.
Type: `RootKeyStorage`.

**FVP_Base_Cortex_A77.bp.trusted_nv_counter**

Trusted Non-Volatile Counter unit.
Type: `NonVolatileCounter`.

**FVP_Base_Cortex_A77.bp.trusted_rng**

Random Number Generator unit.
Type: `RandomNumberGenerator`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A77.bp.trusted_watchdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A77.bp.trusted_watchdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A77.bp.trusted_watchdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A77.bp.trusted_watchdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A77.bp.trusted_watchdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A77.bp.tzc_400**

TrustZone Address Space Controller.
Type: `TZC_400`.

**FVP_Base_Cortex_A77.bp.ve_sysregs**

Type: `VE_SysRegs`.

**FVP_Base_Cortex_A77.bp.vedcc**

Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

**FVP_Base_Cortex_A77.bp.virtio_net**

VirtioNet device over MMIO transport.
Type: `VirtioNetMMIO`.

**FVP_Base_Cortex_A77.bp.virtio_net_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A77.bp.virtio_rng**

VirtioEntropy device over MMIO transport.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `VirtioEntropyMMIO`.

**FVP_Base_Cortex_A77.bp.virtioblockdevice**

VirtioBlock device over MMIO transport.
Type: `VirtioBlockMMIO`.

**FVP_Base_Cortex_A77.bp.virtioblockdevice_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A77.bp.virtiop9device**

virtio P9 server.
Type: `VirtioP9Device`.

**FVP_Base_Cortex_A77.bp.virtiop9device_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A77.bp.vis**

Display window for VE using Visualisation library.
Type: `VEVisualisation`.

**FVP_Base_Cortex_A77.bp.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_Base_Cortex_A77.bp.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A77.bp.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A77.bp.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A77.bp.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A77.bp.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A77.bp.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A77.bp.vram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A77.cci400**

Cache Coherent Interconnect for AXI4 ACE.
Type: `CCI400`.

**FVP_Base_Cortex_A77.clockdivider0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A77.cluster0**

ARM Cortex-A77 Cluster CT model.
Type: `Cluster_ARM_Cortex-A77`.

**FVP_Base_Cortex_A77.cluster0.DSU.shared_cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A77.cluster0.cpu0**

ARM Cortex-A77 CT model.
Type: `ARM_Cortex-A77`.

**FVP_Base_Cortex_A77.cluster0.cpu0.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A77.cluster0.cpu0.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A77.cluster0.cpu0.l1icache**

PV Cache.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `PVCache`.

**FVP_Base_Cortex_A77.cluster0.cpu0.l2cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A77.cluster0.cpu1**

ARM Cortex-A77 CT model.
Type: `ARM_Cortex-A77`.

**FVP_Base_Cortex_A77.cluster0.cpu1.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A77.cluster0.cpu1.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A77.cluster0.cpu1.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A77.cluster0.cpu1.l2cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A77.cluster0.cpu2**

ARM Cortex-A77 CT model.
Type: `ARM_Cortex-A77`.

**FVP_Base_Cortex_A77.cluster0.cpu2.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A77.cluster0.cpu2.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A77.cluster0.cpu2.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A77.cluster0.cpu2.l2cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A77.cluster0.cpu3**

ARM Cortex-A77 CT model.
Type: `ARM_Cortex-A77`.

**FVP_Base_Cortex_A77.cluster0.cpu3.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A77.cluster0.cpu3.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A77.cluster0.cpu3.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A77.cluster0.cpu3.l2cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A77.cluster0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A77.dapmemlogger**

Bus Logger.
Type: `PVBusLogger`.

**FVP_Base_Cortex_A77.elfloader**

ELF loader component.
Type: `ElfLoader`.

**FVP_Base_Cortex_A77.gic_distributor**

GIC Interrupt Redistribution Infrastucture component.
Type: `GIC_IRI`.

**FVP_Base_Cortex_A77.pctl**

Base Platforms Power Controller.
Type: `Base_PowerController`.

**FVP_Base_Cortex_A77.pctl.timer_reset**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A77.pctl.timer_reset.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A77.pctl.timer_reset.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `SchedulerThread`.

**`FVP_Base_Cortex_A77.pctl.timer_reset.timer.thread_event`**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

# 13.60 FVP_Base_Cortex-A78

List of instances in FVP_Base_Cortex-A78.

## FVP_Base_Cortex-A78 instances

**`FVP_Base_Cortex_A78`**

Base Platform Compute Subsystem for ARMCortexA78CT.
Type: `FVP_Base_Cortex_A78`.

**`FVP_Base_Cortex_A78.bp`**

Peripherals and address map for the Base Platform.
Type: `BasePlatformPeripherals`.

**`FVP_Base_Cortex_A78.bp.Timer_0_1`**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**`FVP_Base_Cortex_A78.bp.Timer_0_1.clk_div0`**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**`FVP_Base_Cortex_A78.bp.Timer_0_1.clk_div1`**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**`FVP_Base_Cortex_A78.bp.Timer_0_1.counter0`**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**`FVP_Base_Cortex_A78.bp.Timer_0_1.counter1`**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**`FVP_Base_Cortex_A78.bp.Timer_2_3`**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A78.bp.Timer_2_3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A78.bp.Timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A78.bp.Timer_2_3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A78.bp.Timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A78.bp.ap_refclk**

ARM Generic Timer.
Type: `MemoryMappedGenericTimer`.

**FVP_Base_Cortex_A78.bp.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**FVP_Base_Cortex_A78.bp.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A78.bp.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A78.bp.clock300MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A78.bp.clock32KHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A78.bp.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A78.bp.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A78.bp.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A78.bp.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A78.bp.dmc**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A78.bp.dmc_phy**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A78.bp.dummy_local_dap_rom**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A78.bp.dummy_ram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A78.bp.dummy_usb**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A78.bp.exclusive_monitor**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_Base_Cortex_A78.bp.flash0**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A78.bp.flash1**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A78.bp.flashloader0**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A78.bp.flashloader1**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A78.bp.generic_watchdog**

ARM Generic Watchdog.
Type: `MemoryMappedGenericWatchdog`.

**FVP_Base_Cortex_A78.bp.hdlcd0**

ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).
Type: `PL370_HDLCD`.

**FVP_Base_Cortex_A78.bp.hdlcd0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A78.bp.hdlcd0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A78.bp.hdlcd0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A78.bp.hdlcd0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A78.bp.hdlcd0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A78.bp.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_Base_Cortex_A78.bp.mmc**

Generic Multimedia Card.
Type: `MMC`.

**FVP_Base_Cortex_A78.bp.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A78.bp.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A78.bp.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A78.bp.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A78.bp.mpe**

Encrypt memory transactions for each encryption context with an independent key to prevent mismatch access.
Type: `PVMemoryProtectionEngine`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A78.bp.nontrustedrom**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A78.bp.nontrustedromloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A78.bp.ns_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A78.bp.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A78.bp.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A78.bp.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A78.bp.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A78.bp.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A78.bp.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A78.bp.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A78.bp.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A78.bp.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A78.bp.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A78.bp.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A78.bp.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A78.bp.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A78.bp.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A78.bp.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A78.bp.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A78.bp.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A78.bp.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A78.bp.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A78.bp.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A78.bp.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A78.bp.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A78.bp.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A78.bp.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A78.bp.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

**FVP_Base_Cortex_A78.bp.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A78.bp.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A78.bp.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A78.bp.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A78.bp.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_Base_Cortex_A78.bp.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A78.bp.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A78.bp.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A78.bp.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A78.bp.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A78.bp.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A78.bp.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A78.bp.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A78.bp.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_Base_Cortex_A78.bp.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

**FVP_Base_Cortex_A78.bp.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A78.bp.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A78.bp.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A78.bp.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A78.bp.pl111_clcd_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A78.bp.pl180_mci**

ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

**FVP_Base_Cortex_A78.bp.ps2keyboard**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A78.bp.ps2keyboard.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A78.bp.ps2keyboard.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A78.bp.ps2keyboard.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A78.bp.ps2keyboard.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A78.bp.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

**FVP_Base_Cortex_A78.bp.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A78.bp.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A78.bp.ps2mouse.ps2_clocktimer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A78.bp.ps2mouse.ps2_clocktimer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A78.bp.psram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A78.bp.refcounter**
Memory Mapped Counter Module for Generic Timers.
Type: `MemoryMappedCounterModule`.

**FVP_Base_Cortex_A78.bp.reset_or**
Or Gate.
Type: `OrGate`.

**FVP_Base_Cortex_A78.bp.rl_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A78.bp.rt_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A78.bp.s_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A78.bp.secureDRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A78.bp.secureSRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A78.bp.secureflash**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A78.bp.secureflashloader**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A78.bp.smsc_91c111**

SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_Base_Cortex_A78.bp.sp805_wdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A78.bp.sp805_wdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A78.bp.sp805_wdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A78.bp.sp805_wdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A78.bp.sp805_wdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A78.bp.sp810_sysctrl**

Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_Base_Cortex_A78.bp.sp810_sysctrl.clkdiv_clk0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A78.bp.sp810_sysctrl.clkdiv_clk1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A78.bp.sp810_sysctrl.clkdiv_clk2**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A78.bp.sp810_sysctrl.clkdiv_clk3**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A78.bp.sram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A78.bp.terminal_0**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A78.bp.terminal_1**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A78.bp.terminal_2**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A78.bp.terminal_3**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A78.bp.trusted_key_storage**

Trusted Root-Key Storage unit.
Type: `RootKeyStorage`.

**FVP_Base_Cortex_A78.bp.trusted_nv_counter**

Trusted Non-Volatile Counter unit.
Type: `NonVolatileCounter`.

**FVP_Base_Cortex_A78.bp.trusted_rng**

Random Number Generator unit.
Type: `RandomNumberGenerator`.

**FVP_Base_Cortex_A78.bp.trusted_watchdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A78.bp.trusted_watchdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A78.bp.trusted_watchdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A78.bp.trusted_watchdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A78.bp.trusted_watchdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A78.bp.tzc_400**

TrustZone Address Space Controller.
Type: `TZC_400`.

**FVP_Base_Cortex_A78.bp.ve_sysregs**

Type: `VE_SysRegs`.

**FVP_Base_Cortex_A78.bp.vedcc**

Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

**FVP_Base_Cortex_A78.bp.virtio_net**

VirtioNet device over MMIO transport.
Type: `VirtioNetMMIO`.

**FVP_Base_Cortex_A78.bp.virtio_net_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A78.bp.virtio_rng**

VirtioEntropy device over MMIO transport.
Type: `VirtioEntropyMMIO`.

**FVP_Base_Cortex_A78.bp.virtioblockdevice**

VirtioBlock device over MMIO transport.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `VirtioBlockMMIO`.

**FVP_Base_Cortex_A78.bp.virtioblockdevice_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A78.bp.virtiop9device**

virtio P9 server.
Type: `VirtioP9Device`.

**FVP_Base_Cortex_A78.bp.virtiop9device_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A78.bp.vis**

Display window for VE using Visualisation library.
Type: `VEVisualisation`.

**FVP_Base_Cortex_A78.bp.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_Base_Cortex_A78.bp.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A78.bp.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A78.bp.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A78.bp.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A78.bp.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A78.bp.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A78.bp.vram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A78.cci400**

Cache Coherent Interconnect for AXI4 ACE.
Type: `CCI400`.

**FVP_Base_Cortex_A78.clockdivider0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A78.cluster0**

ARM Cortex-A78 Cluster CT model.
Type: `Cluster_ARM_Cortex-A78`.

**FVP_Base_Cortex_A78.cluster0.DSU.shared_cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A78.cluster0.cpu0**

ARM Cortex-A78 CT model.
Type: `ARM_Cortex-A78`.

**FVP_Base_Cortex_A78.cluster0.cpu0.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A78.cluster0.cpu0.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A78.cluster0.cpu0.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A78.cluster0.cpu0.l2cache**

PV Cache.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `PVCache`.

**`FVP_Base_Cortex_A78.cluster0.cpu1`**

ARM Cortex-A78 CT model.
Type: `ARM_Cortex-A78`.

**`FVP_Base_Cortex_A78.cluster0.cpu1.dtlb`**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**`FVP_Base_Cortex_A78.cluster0.cpu1.l1dcache`**

PV Cache.
Type: `PVCache`.

**`FVP_Base_Cortex_A78.cluster0.cpu1.l1icache`**

PV Cache.
Type: `PVCache`.

**`FVP_Base_Cortex_A78.cluster0.cpu1.l2cache`**

PV Cache.
Type: `PVCache`.

**`FVP_Base_Cortex_A78.cluster0.cpu2`**

ARM Cortex-A78 CT model.
Type: `ARM_Cortex-A78`.

**`FVP_Base_Cortex_A78.cluster0.cpu2.dtlb`**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**`FVP_Base_Cortex_A78.cluster0.cpu2.l1dcache`**

PV Cache.
Type: `PVCache`.

**`FVP_Base_Cortex_A78.cluster0.cpu2.l1icache`**

PV Cache.
Type: `PVCache`.

**`FVP_Base_Cortex_A78.cluster0.cpu2.l2cache`**

PV Cache.
Type: `PVCache`.

**`FVP_Base_Cortex_A78.cluster0.cpu3`**

ARM Cortex-A78 CT model.
Type: `ARM_Cortex-A78`.

**`FVP_Base_Cortex_A78.cluster0.cpu3.dtlb`**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**`FVP_Base_Cortex_A78.cluster0.cpu3.l1dcache`**

PV Cache.
Type: `PVCache`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A78.cluster0.cpu3.l1icache**

PV Cache.
Type: PVCache.

**FVP_Base_Cortex_A78.cluster0.cpu3.l2cache**

PV Cache.
Type: PVCache.

**FVP_Base_Cortex_A78.cluster0_labeller**

Type: Labeller.

**FVP_Base_Cortex_A78.dapmemlogger**

Bus Logger.
Type: PVBusLogger.

**FVP_Base_Cortex_A78.elfloader**

ELF loader component.
Type: ElfLoader.

**FVP_Base_Cortex_A78.gic_distributor**

GIC Interrupt Redistribution Infrastucture component.
Type: GIC_IRI.

**FVP_Base_Cortex_A78.pctl**

Base Platforms Power Controller.
Type: Base_PowerController.

**FVP_Base_Cortex_A78.pctl.timer_reset**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: ClockTimerThread.

**FVP_Base_Cortex_A78.pctl.timer_reset.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: ClockTimerThread64.

**FVP_Base_Cortex_A78.pctl.timer_reset.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: SchedulerThread.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A78.pctl.timer_reset.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

# 13.61 FVP_Base_Cortex-A78AE

List of instances in FVP_Base_Cortex-A78AE.

## FVP_Base_Cortex-A78AE instances

**FVP_Base_Cortex_A78AE**

Base Platform Compute Subsystem for ARMCortexA78AECT.
Type: `FVP_Base_Cortex_A78AE`.

**FVP_Base_Cortex_A78AE.bp**

Peripherals and address map for the Base Platform.
Type: `BasePlatformPeripherals`.

**FVP_Base_Cortex_A78AE.bp.Timer_0_1**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Cortex_A78AE.bp.Timer_0_1.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A78AE.bp.Timer_0_1.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A78AE.bp.Timer_0_1.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A78AE.bp.Timer_0_1.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A78AE.bp.Timer_2_3**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A78AE.bp.Timer_2_3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A78AE.bp.Timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A78AE.bp.Timer_2_3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A78AE.bp.Timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A78AE.bp.ap_refclk**

ARM Generic Timer.
Type: `MemoryMappedGenericTimer`.

**FVP_Base_Cortex_A78AE.bp.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**FVP_Base_Cortex_A78AE.bp.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A78AE.bp.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A78AE.bp.clock300MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A78AE.bp.clock32KHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A78AE.bp.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A78AE.bp.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A78AE.bp.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A78AE.bp.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A78AE.bp.dmc**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A78AE.bp.dmc_phy**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A78AE.bp.dummy_local_dap_rom**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A78AE.bp.dummy_ram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A78AE.bp.dummy_usb**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A78AE.bp.exclusive_monitor**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_Base_Cortex_A78AE.bp.flash0**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A78AE.bp.flash1**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A78AE.bp.flashloader0**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A78AE.bp.flashloader1**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A78AE.bp.generic_watchdog**

ARM Generic Watchdog.
Type: `MemoryMappedGenericWatchdog`.

**FVP_Base_Cortex_A78AE.bp.hdlcd0**

ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).
Type: `PL370_HDLCD`.

**FVP_Base_Cortex_A78AE.bp.hdlcd0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A78AE.bp.hdlcd0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A78AE.bp.hdlcd0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A78AE.bp.hdlcd0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A78AE.bp.hdlcd0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A78AE.bp.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_Base_Cortex_A78AE.bp.mmc**

Generic Multimedia Card.
Type: `MMC`.

**FVP_Base_Cortex_A78AE.bp.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A78AE.bp.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A78AE.bp.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A78AE.bp.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A78AE.bp.mpe**

Encrypt memory transactions for each encryption context with an independent key to prevent mismatch access.
Type: `PVMemoryProtectionEngine`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A78AE.bp.nontrustedrom**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A78AE.bp.nontrustedromloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A78AE.bp.ns_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A78AE.bp.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A78AE.bp.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A78AE.bp.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A78AE.bp.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A78AE.bp.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A78AE.bp.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A78AE.bp.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A78AE.bp.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A78AE.bp.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A78AE.bp.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A78AE.bp.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A78AE.bp.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A78AE.bp.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A78AE.bp.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A78AE.bp.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A78AE.bp.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A78AE.bp.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A78AE.bp.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A78AE.bp.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A78AE.bp.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A78AE.bp.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A78AE.bp.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A78AE.bp.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A78AE.bp.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A78AE.bp.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

**FVP_Base_Cortex_A78AE.bp.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A78AE.bp.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A78AE.bp.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A78AE.bp.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A78AE.bp.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_Base_Cortex_A78AE.bp.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A78AE.bp.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A78AE.bp.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A78AE.bp.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A78AE.bp.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A78AE.bp.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A78AE.bp.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A78AE.bp.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A78AE.bp.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_Base_Cortex_A78AE.bp.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

**FVP_Base_Cortex_A78AE.bp.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A78AE.bp.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A78AE.bp.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A78AE.bp.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A78AE.bp.pl111_clcd_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A78AE.bp.pl180_mci**

ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

**FVP_Base_Cortex_A78AE.bp.ps2keyboard**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A78AE.bp.ps2keyboard.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A78AE.bp.ps2keyboard.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A78AE.bp.ps2keyboard.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A78AE.bp.ps2keyboard.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A78AE.bp.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

**FVP_Base_Cortex_A78AE.bp.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A78AE.bp.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A78AE.bp.ps2mouse.ps2_clocktimer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A78AE.bp.ps2mouse.ps2_clocktimer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A78AE.bp.psram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A78AE.bp.refcounter**
Memory Mapped Counter Module for Generic Timers.
Type: `MemoryMappedCounterModule`.

**FVP_Base_Cortex_A78AE.bp.reset_or**
Or Gate.
Type: `OrGate`.

**FVP_Base_Cortex_A78AE.bp.rl_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A78AE.bp.rt_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A78AE.bp.s_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A78AE.bp.secureDRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A78AE.bp.secureSRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A78AE.bp.secureflash**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A78AE.bp.secureflashloader**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A78AE.bp.smsc_91c111**

SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_Base_Cortex_A78AE.bp.sp805_wdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A78AE.bp.sp805_wdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A78AE.bp.sp805_wdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A78AE.bp.sp805_wdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A78AE.bp.sp805_wdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A78AE.bp.sp810_sysctrl**

Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_Base_Cortex_A78AE.bp.sp810_sysctrl.clkdiv_clk0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A78AE.bp.sp810_sysctrl.clkdiv_clk1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A78AE.bp.sp810_sysctrl.clkdiv_clk2**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A78AE.bp.sp810_sysctrl.clkdiv_clk3**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A78AE.bp.sram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A78AE.bp.terminal_0**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A78AE.bp.terminal_1**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A78AE.bp.terminal_2**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A78AE.bp.terminal_3**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A78AE.bp.trusted_key_storage**
Trusted Root-Key Storage unit.
Type: `RootKeyStorage`.

**FVP_Base_Cortex_A78AE.bp.trusted_nv_counter**
Trusted Non-Volatile Counter unit.
Type: `NonVolatileCounter`.

**FVP_Base_Cortex_A78AE.bp.trusted_rng**
Random Number Generator unit.
Type: `RandomNumberGenerator`.

**FVP_Base_Cortex_A78AE.bp.trusted_watchdog**
ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A78AE.bp.trusted_watchdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A78AE.bp.trusted_watchdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A78AE.bp.trusted_watchdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A78AE.bp.trusted_watchdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A78AE.bp.tzc_400**

TrustZone Address Space Controller.
Type: `TZC_400`.

**FVP_Base_Cortex_A78AE.bp.ve_sysregs**

Type: `VE_SysRegs`.

**FVP_Base_Cortex_A78AE.bp.vedcc**

Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

**FVP_Base_Cortex_A78AE.bp.virtio_net**

VirtioNet device over MMIO transport.
Type: `VirtioNetMMIO`.

**FVP_Base_Cortex_A78AE.bp.virtio_net_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A78AE.bp.virtio_rng**

VirtioEntropy device over MMIO transport.
Type: `VirtioEntropyMMIO`.

**FVP_Base_Cortex_A78AE.bp.virtioblockdevice**

VirtioBlock device over MMIO transport.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `VirtioBlockMMIO`.

**FVP_Base_Cortex_A78AE.bp.virtioblockdevice_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A78AE.bp.virtiop9device**

virtio P9 server.
Type: `VirtioP9Device`.

**FVP_Base_Cortex_A78AE.bp.virtiop9device_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A78AE.bp.vis**

Display window for VE using Visualisation library.
Type: `VEVisualisation`.

**FVP_Base_Cortex_A78AE.bp.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_Base_Cortex_A78AE.bp.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A78AE.bp.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A78AE.bp.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A78AE.bp.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A78AE.bp.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A78AE.bp.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A78AE.bp.vram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A78AE.cci400**

Cache Coherent Interconnect for AXI4 ACE.
Type: `CCI400`.

**FVP_Base_Cortex_A78AE.clockdivider0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A78AE.cluster0**

ARM Cortex-A78AE Cluster CT model.
Type: `Cluster_ARM_Cortex-A78AE`.

**FVP_Base_Cortex_A78AE.cluster0.DSU.shared_cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A78AE.cluster0.cpu0**

ARM Cortex-A78AE CT model.
Type: `ARM_Cortex-A78AE`.

**FVP_Base_Cortex_A78AE.cluster0.cpu0.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A78AE.cluster0.cpu0.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A78AE.cluster0.cpu0.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A78AE.cluster0.cpu0.l2cache**

PV Cache.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `PVCache`.

**FVP_Base_Cortex_A78AE.cluster0.cpu1**

ARM Cortex-A78AE CT model.
Type: `ARM_Cortex-A78AE`.

**FVP_Base_Cortex_A78AE.cluster0.cpu1.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A78AE.cluster0.cpu1.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A78AE.cluster0.cpu1.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A78AE.cluster0.cpu1.l2cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A78AE.cluster0.cpu2**

ARM Cortex-A78AE CT model.
Type: `ARM_Cortex-A78AE`.

**FVP_Base_Cortex_A78AE.cluster0.cpu2.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A78AE.cluster0.cpu2.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A78AE.cluster0.cpu2.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A78AE.cluster0.cpu2.l2cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A78AE.cluster0.cpu3**

ARM Cortex-A78AE CT model.
Type: `ARM_Cortex-A78AE`.

**FVP_Base_Cortex_A78AE.cluster0.cpu3.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A78AE.cluster0.cpu3.l1dcache**

PV Cache.
Type: `PVCache`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A78AE.cluster0.cpu3.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A78AE.cluster0.cpu3.l2cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A78AE.cluster0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A78AE.dapmemlogger**

Bus Logger.
Type: `PVBusLogger`.

**FVP_Base_Cortex_A78AE.elfloader**

ELF loader component.
Type: `ElfLoader`.

**FVP_Base_Cortex_A78AE.gic_distributor**

GIC Interrupt Redistribution Infrastucture component.
Type: `GIC_IRI`.

**FVP_Base_Cortex_A78AE.pctl**

Base Platforms Power Controller.
Type: `Base_PowerController`.

**FVP_Base_Cortex_A78AE.pctl.timer_reset**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A78AE.pctl.timer_reset.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A78AE.pctl.timer_reset.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A78AE.pctl.timer_reset.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

## 13.62 FVP_Base_Cortex-A78C

List of instances in FVP_Base_Cortex-A78C.

### FVP_Base_Cortex-A78C instances

**FVP_Base_Cortex_A78C**

Base Platform Compute Subsystem for ARMCortexA78CCT.
Type: `FVP_Base_Cortex_A78C`.

**FVP_Base_Cortex_A78C.bp**

Peripherals and address map for the Base Platform.
Type: `BasePlatformPeripherals`.

**FVP_Base_Cortex_A78C.bp.Timer_0_1**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Cortex_A78C.bp.Timer_0_1.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A78C.bp.Timer_0_1.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A78C.bp.Timer_0_1.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A78C.bp.Timer_0_1.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A78C.bp.Timer_2_3**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A78C.bp.Timer_2_3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A78C.bp.Timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A78C.bp.Timer_2_3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A78C.bp.Timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_A78C.bp.ap_refclk**

ARM Generic Timer.
Type: `MemoryMappedGenericTimer`.

**FVP_Base_Cortex_A78C.bp.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**FVP_Base_Cortex_A78C.bp.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A78C.bp.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A78C.bp.clock300MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A78C.bp.clock32KHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A78C.bp.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A78C.bp.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A78C.bp.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A78C.bp.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A78C.bp.dmc**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A78C.bp.dmc_phy**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A78C.bp.dummy_local_dap_rom**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A78C.bp.dummy_ram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A78C.bp.dummy_usb**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A78C.bp.exclusive_monitor**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_Base_Cortex_A78C.bp.flash0**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A78C.bp.flash1**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A78C.bp.flashloader0**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A78C.bp.flashloader1**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A78C.bp.generic_watchdog**

ARM Generic Watchdog.
Type: `MemoryMappedGenericWatchdog`.

**FVP_Base_Cortex_A78C.bp.hdlcd0**

ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).
Type: `PL370_HDLCD`.

**FVP_Base_Cortex_A78C.bp.hdlcd0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A78C.bp.hdlcd0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A78C.bp.hdlcd0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A78C.bp.hdlcd0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A78C.bp.hdlcd0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A78C.bp.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_Base_Cortex_A78C.bp.mmc**

Generic Multimedia Card.
Type: `MMC`.

**FVP_Base_Cortex_A78C.bp.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A78C.bp.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A78C.bp.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A78C.bp.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A78C.bp.mpe**

Encrypt memory transactions for each encryption context with an independent key to prevent mismatch access.
Type: `PVMemoryProtectionEngine`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A78C.bp.nontrustedrom**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A78C.bp.nontrustedromloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_A78C.bp.ns_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A78C.bp.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A78C.bp.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A78C.bp.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A78C.bp.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A78C.bp.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A78C.bp.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A78C.bp.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A78C.bp.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A78C.bp.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A78C.bp.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A78C.bp.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A78C.bp.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A78C.bp.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A78C.bp.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A78C.bp.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A78C.bp.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A78C.bp.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A78C.bp.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A78C.bp.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_A78C.bp.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A78C.bp.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A78C.bp.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A78C.bp.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A78C.bp.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A78C.bp.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

**FVP_Base_Cortex_A78C.bp.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A78C.bp.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A78C.bp.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A78C.bp.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A78C.bp.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_Base_Cortex_A78C.bp.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A78C.bp.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A78C.bp.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A78C.bp.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A78C.bp.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A78C.bp.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A78C.bp.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_A78C.bp.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A78C.bp.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_Base_Cortex_A78C.bp.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

**FVP_Base_Cortex_A78C.bp.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A78C.bp.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A78C.bp.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A78C.bp.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A78C.bp.pl111_clcd_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A78C.bp.pl180_mci**

ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

**FVP_Base_Cortex_A78C.bp.ps2keyboard**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A78C.bp.ps2keyboard.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A78C.bp.ps2keyboard.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A78C.bp.ps2keyboard.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A78C.bp.ps2keyboard.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A78C.bp.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

**FVP_Base_Cortex_A78C.bp.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A78C.bp.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A78C.bp.ps2mouse.ps2_clocktimer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A78C.bp.ps2mouse.ps2_clocktimer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A78C.bp.psram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A78C.bp.refcounter**
Memory Mapped Counter Module for Generic Timers.
Type: `MemoryMappedCounterModule`.

**FVP_Base_Cortex_A78C.bp.reset_or**
Or Gate.
Type: `OrGate`.

**FVP_Base_Cortex_A78C.bp.rl_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A78C.bp.rt_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A78C.bp.s_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A78C.bp.secureDRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A78C.bp.secureSRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A78C.bp.secureflash**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_A78C.bp.secureflashloader**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A78C.bp.smsc_91c111**

SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_Base_Cortex_A78C.bp.sp805_wdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_A78C.bp.sp805_wdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A78C.bp.sp805_wdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A78C.bp.sp805_wdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A78C.bp.sp805_wdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A78C.bp.sp810_sysctrl**

Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_Base_Cortex_A78C.bp.sp810_sysctrl.clkdiv_clk0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A78C.bp.sp810_sysctrl.clkdiv_clk1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A78C.bp.sp810_sysctrl.clkdiv_clk2**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A78C.bp.sp810_sysctrl.clkdiv_clk3**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A78C.bp.sram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A78C.bp.terminal_0**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A78C.bp.terminal_1**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A78C.bp.terminal_2**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A78C.bp.terminal_3**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_A78C.bp.trusted_key_storage**

Trusted Root-Key Storage unit.
Type: `RootKeyStorage`.

**FVP_Base_Cortex_A78C.bp.trusted_nv_counter**

Trusted Non-Volatile Counter unit.
Type: `NonVolatileCounter`.

**FVP_Base_Cortex_A78C.bp.trusted_rng**

Random Number Generator unit.
Type: `RandomNumberGenerator`.

**FVP_Base_Cortex_A78C.bp.trusted_watchdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A78C.bp.trusted_watchdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A78C.bp.trusted_watchdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A78C.bp.trusted_watchdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_A78C.bp.trusted_watchdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A78C.bp.tzc_400**

TrustZone Address Space Controller.
Type: `TZC_400`.

**FVP_Base_Cortex_A78C.bp.ve_sysregs**

Type: `VE_SysRegs`.

**FVP_Base_Cortex_A78C.bp.vedcc**

Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

**FVP_Base_Cortex_A78C.bp.virtio_net**

VirtioNet device over MMIO transport.
Type: `VirtioNetMMIO`.

**FVP_Base_Cortex_A78C.bp.virtio_net_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A78C.bp.virtio_rng**

VirtioEntropy device over MMIO transport.
Type: `VirtioEntropyMMIO`.

**FVP_Base_Cortex_A78C.bp.virtioblockdevice**

VirtioBlock device over MMIO transport.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `VirtioBlockMMIO`.

**FVP_Base_Cortex_A78C.bp.virtioblockdevice_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A78C.bp.virtiop9device**

virtio P9 server.
Type: `VirtioP9Device`.

**FVP_Base_Cortex_A78C.bp.virtiop9device_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A78C.bp.vis**

Display window for VE using Visualisation library.
Type: `VEVisualisation`.

**FVP_Base_Cortex_A78C.bp.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_Base_Cortex_A78C.bp.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A78C.bp.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A78C.bp.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A78C.bp.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A78C.bp.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_A78C.bp.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A78C.bp.vram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_A78C.cci400**

Cache Coherent Interconnect for AXI4 ACE.
Type: `CCI400`.

**FVP_Base_Cortex_A78C.clockdivider0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_A78C.cluster0**

ARM Cortex-A78C Cluster CT model.
Type: `Cluster_ARM_Cortex-A78C`.

**FVP_Base_Cortex_A78C.cluster0.DSU.shared_cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A78C.cluster0.cpu0**

ARM Cortex-A78C CT model.
Type: `ARM_Cortex-A78C`.

**FVP_Base_Cortex_A78C.cluster0.cpu0.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A78C.cluster0.cpu0.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A78C.cluster0.cpu0.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A78C.cluster0.cpu0.l2cache**

PV Cache.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `PVCache`.

**FVP_Base_Cortex_A78C.cluster0.cpu1**

ARM Cortex-A78C CT model.
Type: `ARM_Cortex-A78C`.

**FVP_Base_Cortex_A78C.cluster0.cpu1.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A78C.cluster0.cpu1.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A78C.cluster0.cpu1.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A78C.cluster0.cpu1.l2cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A78C.cluster0.cpu2**

ARM Cortex-A78C CT model.
Type: `ARM_Cortex-A78C`.

**FVP_Base_Cortex_A78C.cluster0.cpu2.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A78C.cluster0.cpu2.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A78C.cluster0.cpu2.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A78C.cluster0.cpu2.l2cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A78C.cluster0.cpu3**

ARM Cortex-A78C CT model.
Type: `ARM_Cortex-A78C`.

**FVP_Base_Cortex_A78C.cluster0.cpu3.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_A78C.cluster0.cpu3.l1dcache**

PV Cache.
Type: `PVCache`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A78C.cluster0.cpu3.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A78C.cluster0.cpu3.l2cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_A78C.cluster0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_A78C.dapmemlogger**

Bus Logger.
Type: `PVBusLogger`.

**FVP_Base_Cortex_A78C.elfloader**

ELF loader component.
Type: `ElfLoader`.

**FVP_Base_Cortex_A78C.gic_distributor**

GIC Interrupt Redistribution Infrastucture component.
Type: `GIC_IRI`.

**FVP_Base_Cortex_A78C.pctl**

Base Platforms Power Controller.
Type: `Base_PowerController`.

**FVP_Base_Cortex_A78C.pctl.timer_reset**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_A78C.pctl.timer_reset.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_A78C.pctl.timer_reset.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_A78C.pctl.timer_reset.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

# 13.63 FVP_Base_Cortex-X1

List of instances in FVP_Base_Cortex-X1.

## FVP_Base_Cortex-X1 instances

**FVP_Base_Cortex_X1**

Base Platform Compute Subsystem for ARMCortexX1CT.
Type: `FVP_Base_Cortex_X1`.

**FVP_Base_Cortex_X1.bp**

Peripherals and address map for the Base Platform.
Type: `BasePlatformPeripherals`.

**FVP_Base_Cortex_X1.bp.Timer_0_1**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Cortex_X1.bp.Timer_0_1.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X1.bp.Timer_0_1.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X1.bp.Timer_0_1.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_X1.bp.Timer_0_1.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_X1.bp.Timer_2_3**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_X1.bp.Timer_2_3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X1.bp.Timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X1.bp.Timer_2_3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_X1.bp.Timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_X1.bp.ap_refclk**

ARM Generic Timer.
Type: `MemoryMappedGenericTimer`.

**FVP_Base_Cortex_X1.bp.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**FVP_Base_Cortex_X1.bp.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X1.bp.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X1.bp.clock300MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_X1.bp.clock32KHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X1.bp.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X1.bp.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X1.bp.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X1.bp.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X1.bp.dmc**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_X1.bp.dmc_phy**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_X1.bp.dummy_local_dap_rom**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_X1.bp.dummy_ram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_X1.bp.dummy_usb**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_X1.bp.exclusive_monitor**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_Base_Cortex_X1.bp.flash0**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_X1.bp.flash1**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_X1.bp.flashloader0**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_X1.bp.flashloader1**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_X1.bp.generic_watchdog**

ARM Generic Watchdog.
Type: `MemoryMappedGenericWatchdog`.

**FVP_Base_Cortex_X1.bp.hdlcd0**

ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).
Type: `PL370_HDLCD`.

**FVP_Base_Cortex_X1.bp.hdlcd0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_X1.bp.hdlcd0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_X1.bp.hdlcd0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_X1.bp.hdlcd0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_X1.bp.hdlcd0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_X1.bp.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_Base_Cortex_X1.bp.mmc**

Generic Multimedia Card.
Type: `MMC`.

**FVP_Base_Cortex_X1.bp.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_X1.bp.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_X1.bp.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_X1.bp.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_X1.bp.mpe**

Encrypt memory transactions for each encryption context with an independent key to prevent mismatch access.
Type: `PVMemoryProtectionEngine`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_X1.bp.nontrustedrom**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_X1.bp.nontrustedromloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_X1.bp.ns_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_X1.bp.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_X1.bp.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X1.bp.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_X1.bp.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_X1.bp.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_X1.bp.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_X1.bp.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_X1.bp.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X1.bp.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_X1.bp.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_X1.bp.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_X1.bp.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_X1.bp.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_X1.bp.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_X1.bp.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_X1.bp.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_X1.bp.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_X1.bp.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_X1.bp.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_X1.bp.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X1.bp.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_X1.bp.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_X1.bp.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_X1.bp.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_X1.bp.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

**FVP_Base_Cortex_X1.bp.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_X1.bp.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_X1.bp.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_X1.bp.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_X1.bp.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_Base_Cortex_X1.bp.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_X1.bp.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_X1.bp.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_X1.bp.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_X1.bp.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_X1.bp.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X1.bp.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_X1.bp.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X1.bp.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_Base_Cortex_X1.bp.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

**FVP_Base_Cortex_X1.bp.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_X1.bp.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_X1.bp.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_X1.bp.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_X1.bp.pl111_clcd_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_X1.bp.pl180_mci**

ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

**FVP_Base_Cortex_X1.bp.ps2keyboard**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_X1.bp.ps2keyboard.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_X1.bp.ps2keyboard.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_X1.bp.ps2keyboard.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_X1.bp.ps2keyboard.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_X1.bp.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

**FVP_Base_Cortex_X1.bp.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_X1.bp.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_X1.bp.ps2mouse.ps2_clocktimer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_X1.bp.ps2mouse.ps2_clocktimer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_X1.bp.psram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_X1.bp.refcounter**
Memory Mapped Counter Module for Generic Timers.
Type: `MemoryMappedCounterModule`.

**FVP_Base_Cortex_X1.bp.reset_or**
Or Gate.
Type: `OrGate`.

**FVP_Base_Cortex_X1.bp.rl_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_X1.bp.rt_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_X1.bp.s_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_X1.bp.secureDRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_X1.bp.secureSRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_X1.bp.secureflash**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_X1.bp.secureflashloader**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_X1.bp.smsc_91c111**

SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_Base_Cortex_X1.bp.sp805_wdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_X1.bp.sp805_wdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_X1.bp.sp805_wdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_X1.bp.sp805_wdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_X1.bp.sp805_wdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_X1.bp.sp810_sysctrl**

Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_Base_Cortex_X1.bp.sp810_sysctrl.clkdiv_clk0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X1.bp.sp810_sysctrl.clkdiv_clk1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X1.bp.sp810_sysctrl.clkdiv_clk2**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X1.bp.sp810_sysctrl.clkdiv_clk3**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X1.bp.sram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_X1.bp.terminal_0**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_X1.bp.terminal_1**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_X1.bp.terminal_2**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_X1.bp.terminal_3**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_X1.bp.trusted_key_storage**

Trusted Root-Key Storage unit.
Type: `RootKeyStorage`.

**FVP_Base_Cortex_X1.bp.trusted_nv_counter**

Trusted Non-Volatile Counter unit.
Type: `NonVolatileCounter`.

**FVP_Base_Cortex_X1.bp.trusted_rng**

Random Number Generator unit.
Type: `RandomNumberGenerator`.

**FVP_Base_Cortex_X1.bp.trusted_watchdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_X1.bp.trusted_watchdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_X1.bp.trusted_watchdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_X1.bp.trusted_watchdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_X1.bp.trusted_watchdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_X1.bp.tzc_400**

TrustZone Address Space Controller.
Type: `TZC_400`.

**FVP_Base_Cortex_X1.bp.ve_sysregs**

Type: `VE_SysRegs`.

**FVP_Base_Cortex_X1.bp.vedcc**

Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

**FVP_Base_Cortex_X1.bp.virtio_net**

VirtioNet device over MMIO transport.
Type: `VirtioNetMMIO`.

**FVP_Base_Cortex_X1.bp.virtio_net_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_X1.bp.virtio_rng**

VirtioEntropy device over MMIO transport.
Type: `VirtioEntropyMMIO`.

**FVP_Base_Cortex_X1.bp.virtioblockdevice**

VirtioBlock device over MMIO transport.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `VirtioBlockMMIO`.

**FVP_Base_Cortex_X1.bp.virtioblockdevice_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_X1.bp.virtiop9device**

virtio P9 server.
Type: `VirtioP9Device`.

**FVP_Base_Cortex_X1.bp.virtiop9device_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_X1.bp.vis**

Display window for VE using Visualisation library.
Type: `VEVisualisation`.

**FVP_Base_Cortex_X1.bp.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_Base_Cortex_X1.bp.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X1.bp.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_X1.bp.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_X1.bp.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_X1.bp.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_X1.bp.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X1.bp.vram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_X1.cci400**

Cache Coherent Interconnect for AXI4 ACE.
Type: `CCI400`.

**FVP_Base_Cortex_X1.clockdivider0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X1.cluster0**

ARM Cortex-X1 Cluster CT model.
Type: `Cluster_ARM_Cortex-X1`.

**FVP_Base_Cortex_X1.cluster0.DSU.shared_cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X1.cluster0.cpu0**

ARM Cortex-X1 CT model.
Type: `ARM_Cortex-X1`.

**FVP_Base_Cortex_X1.cluster0.cpu0.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_X1.cluster0.cpu0.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X1.cluster0.cpu0.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X1.cluster0.cpu0.l2cache**

PV Cache.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `PVCache`.

**FVP_Base_Cortex_X1.cluster0.cpu1**

ARM Cortex-X1 CT model.
Type: `ARM_Cortex-X1`.

**FVP_Base_Cortex_X1.cluster0.cpu1.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_X1.cluster0.cpu1.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X1.cluster0.cpu1.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X1.cluster0.cpu1.l2cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X1.cluster0.cpu2**

ARM Cortex-X1 CT model.
Type: `ARM_Cortex-X1`.

**FVP_Base_Cortex_X1.cluster0.cpu2.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_X1.cluster0.cpu2.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X1.cluster0.cpu2.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X1.cluster0.cpu2.l2cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X1.cluster0.cpu3**

ARM Cortex-X1 CT model.
Type: `ARM_Cortex-X1`.

**FVP_Base_Cortex_X1.cluster0.cpu3.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_X1.cluster0.cpu3.l1dcache**

PV Cache.
Type: `PVCache`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_X1.cluster0.cpu3.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X1.cluster0.cpu3.l2cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X1.cluster0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_X1.dapmemlogger**

Bus Logger.
Type: `PVBusLogger`.

**FVP_Base_Cortex_X1.elfloader**

ELF loader component.
Type: `ElfLoader`.

**FVP_Base_Cortex_X1.gic_distributor**

GIC Interrupt Redistribution Infrastucture component.
Type: `GIC_IRI`.

**FVP_Base_Cortex_X1.pctl**

Base Platforms Power Controller.
Type: `Base_PowerController`.

**FVP_Base_Cortex_X1.pctl.timer_reset**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_X1.pctl.timer_reset.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_X1.pctl.timer_reset.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_X1.pctl.timer_reset.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

# 13.64 FVP_Base_Cortex-X1C

List of instances in FVP_Base_Cortex-X1C.

## FVP_Base_Cortex-X1C instances

**FVP_Base_Cortex_X1C**

Base Platform Compute Subsystem for ARMCortexX1CCT.
Type: `FVP_Base_Cortex_X1C`.

**FVP_Base_Cortex_X1C.bp**

Peripherals and address map for the Base Platform.
Type: `BasePlatformPeripherals`.

**FVP_Base_Cortex_X1C.bp.Timer_0_1**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Cortex_X1C.bp.Timer_0_1.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X1C.bp.Timer_0_1.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X1C.bp.Timer_0_1.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_X1C.bp.Timer_0_1.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_X1C.bp.Timer_2_3**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_X1C.bp.Timer_2_3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X1C.bp.Timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X1C.bp.Timer_2_3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_X1C.bp.Timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_X1C.bp.ap_refclk**

ARM Generic Timer.
Type: `MemoryMappedGenericTimer`.

**FVP_Base_Cortex_X1C.bp.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**FVP_Base_Cortex_X1C.bp.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X1C.bp.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X1C.bp.clock300MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_X1C.bp.clock32KHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X1C.bp.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X1C.bp.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X1C.bp.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X1C.bp.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X1C.bp.dmc**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_X1C.bp.dmc_phy**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_X1C.bp.dummy_local_dap_rom**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_X1C.bp.dummy_ram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_X1C.bp.dummy_usb**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_X1C.bp.exclusive_monitor**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_Base_Cortex_X1C.bp.flash0**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_X1C.bp.flash1**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_X1C.bp.flashloader0**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_X1C.bp.flashloader1**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_X1C.bp.generic_watchdog**

ARM Generic Watchdog.
Type: `MemoryMappedGenericWatchdog`.

**FVP_Base_Cortex_X1C.bp.hdlcd0**

ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).
Type: `PL370_HDLCD`.

**FVP_Base_Cortex_X1C.bp.hdlcd0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_X1C.bp.hdlcd0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_X1C.bp.hdlcd0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_X1C.bp.hdlcd0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_X1C.bp.hdlcd0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_X1C.bp.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_Base_Cortex_X1C.bp.mmc**

Generic Multimedia Card.
Type: `MMC`.

**FVP_Base_Cortex_X1C.bp.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_X1C.bp.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_X1C.bp.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_X1C.bp.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_X1C.bp.mpe**

Encrypt memory transactions for each encryption context with an independent key to prevent mismatch access.
Type: `PVMemoryProtectionEngine`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_X1C.bp.nontrustedrom**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_X1C.bp.nontrustedromloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_X1C.bp.ns_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_X1C.bp.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_X1C.bp.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X1C.bp.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_X1C.bp.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_X1C.bp.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_X1C.bp.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_X1C.bp.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_X1C.bp.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X1C.bp.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_X1C.bp.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_X1C.bp.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_X1C.bp.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_X1C.bp.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_X1C.bp.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_X1C.bp.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_X1C.bp.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_X1C.bp.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_X1C.bp.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_X1C.bp.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_X1C.bp.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X1C.bp.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_X1C.bp.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_X1C.bp.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_X1C.bp.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_X1C.bp.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

**FVP_Base_Cortex_X1C.bp.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_X1C.bp.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_X1C.bp.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_X1C.bp.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_X1C.bp.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_Base_Cortex_X1C.bp.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_X1C.bp.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_X1C.bp.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_X1C.bp.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_X1C.bp.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_X1C.bp.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X1C.bp.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_X1C.bp.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X1C.bp.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_Base_Cortex_X1C.bp.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

**FVP_Base_Cortex_X1C.bp.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_X1C.bp.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_X1C.bp.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_X1C.bp.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_X1C.bp.pl111_clcd_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_X1C.bp.pl180_mci**

ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

**FVP_Base_Cortex_X1C.bp.ps2keyboard**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_X1C.bp.ps2keyboard.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_X1C.bp.ps2keyboard.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_X1C.bp.ps2keyboard.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_X1C.bp.ps2keyboard.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_X1C.bp.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

**FVP_Base_Cortex_X1C.bp.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_X1C.bp.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_X1C.bp.ps2mouse.ps2_clocktimer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_X1C.bp.ps2mouse.ps2_clocktimer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_X1C.bp.psram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_X1C.bp.refcounter**
Memory Mapped Counter Module for Generic Timers.
Type: `MemoryMappedCounterModule`.

**FVP_Base_Cortex_X1C.bp.reset_or**
Or Gate.
Type: `OrGate`.

**FVP_Base_Cortex_X1C.bp.rl_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_X1C.bp.rt_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_X1C.bp.s_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_X1C.bp.secureDRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_X1C.bp.secureSRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_X1C.bp.secureflash**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_X1C.bp.secureflashloader**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_X1C.bp.smsc_91c111**

SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_Base_Cortex_X1C.bp.sp805_wdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_X1C.bp.sp805_wdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_X1C.bp.sp805_wdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_X1C.bp.sp805_wdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_X1C.bp.sp805_wdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_X1C.bp.sp810_sysctrl**

Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_Base_Cortex_X1C.bp.sp810_sysctrl.clkdiv_clk0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X1C.bp.sp810_sysctrl.clkdiv_clk1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X1C.bp.sp810_sysctrl.clkdiv_clk2**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X1C.bp.sp810_sysctrl.clkdiv_clk3**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X1C.bp.sram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_X1C.bp.terminal_0**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_X1C.bp.terminal_1**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_X1C.bp.terminal_2**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_X1C.bp.terminal_3**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_X1C.bp.trusted_key_storage**

Trusted Root-Key Storage unit.
Type: `RootKeyStorage`.

**FVP_Base_Cortex_X1C.bp.trusted_nv_counter**

Trusted Non-Volatile Counter unit.
Type: `NonVolatileCounter`.

**FVP_Base_Cortex_X1C.bp.trusted_rng**

Random Number Generator unit.
Type: `RandomNumberGenerator`.

**FVP_Base_Cortex_X1C.bp.trusted_watchdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_X1C.bp.trusted_watchdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_X1C.bp.trusted_watchdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_X1C.bp.trusted_watchdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_X1C.bp.trusted_watchdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_X1C.bp.tzc_400**

TrustZone Address Space Controller.
Type: `TZC_400`.

**FVP_Base_Cortex_X1C.bp.ve_sysregs**

Type: `VE_SysRegs`.

**FVP_Base_Cortex_X1C.bp.vedcc**

Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

**FVP_Base_Cortex_X1C.bp.virtio_net**

VirtioNet device over MMIO transport.
Type: `VirtioNetMMIO`.

**FVP_Base_Cortex_X1C.bp.virtio_net_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_X1C.bp.virtio_rng**

VirtioEntropy device over MMIO transport.
Type: `VirtioEntropyMMIO`.

**FVP_Base_Cortex_X1C.bp.virtioblockdevice**

VirtioBlock device over MMIO transport.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `VirtioBlockMMIO`.

**FVP_Base_Cortex_X1C.bp.virtioblockdevice_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_X1C.bp.virtiop9device**

virtio P9 server.
Type: `VirtioP9Device`.

**FVP_Base_Cortex_X1C.bp.virtiop9device_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_X1C.bp.vis**

Display window for VE using Visualisation library.
Type: `VEVisualisation`.

**FVP_Base_Cortex_X1C.bp.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_Base_Cortex_X1C.bp.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X1C.bp.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_X1C.bp.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_X1C.bp.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_X1C.bp.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_X1C.bp.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X1C.bp.vram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_X1C.cci400**

Cache Coherent Interconnect for AXI4 ACE.
Type: `CCI400`.

**FVP_Base_Cortex_X1C.clockdivider0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X1C.cluster0**

ARM Cortex-X1C Cluster CT model.
Type: `Cluster_ARM_Cortex-X1C`.

**FVP_Base_Cortex_X1C.cluster0.DSU.shared_cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X1C.cluster0.cpu0**

ARM Cortex-X1C CT model.
Type: `ARM_Cortex-X1C`.

**FVP_Base_Cortex_X1C.cluster0.cpu0.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_X1C.cluster0.cpu0.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X1C.cluster0.cpu0.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X1C.cluster0.cpu0.l2cache**

PV Cache.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `PVCache`.

**FVP_Base_Cortex_X1C.cluster0.cpu1**

ARM Cortex-X1C CT model.
Type: `ARM_Cortex-X1C`.

**FVP_Base_Cortex_X1C.cluster0.cpu1.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_X1C.cluster0.cpu1.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X1C.cluster0.cpu1.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X1C.cluster0.cpu1.l2cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X1C.cluster0.cpu2**

ARM Cortex-X1C CT model.
Type: `ARM_Cortex-X1C`.

**FVP_Base_Cortex_X1C.cluster0.cpu2.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_X1C.cluster0.cpu2.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X1C.cluster0.cpu2.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X1C.cluster0.cpu2.l2cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X1C.cluster0.cpu3**

ARM Cortex-X1C CT model.
Type: `ARM_Cortex-X1C`.

**FVP_Base_Cortex_X1C.cluster0.cpu3.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_X1C.cluster0.cpu3.l1dcache**

PV Cache.
Type: `PVCache`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_X1C.cluster0.cpu3.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X1C.cluster0.cpu3.l2cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X1C.cluster0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_X1C.dapmemlogger**

Bus Logger.
Type: `PVBusLogger`.

**FVP_Base_Cortex_X1C.elfloader**

ELF loader component.
Type: `ElfLoader`.

**FVP_Base_Cortex_X1C.gic_distributor**

GIC Interrupt Redistribution Infrastucture component.
Type: `GIC_IRI`.

**FVP_Base_Cortex_X1C.pctl**

Base Platforms Power Controller.
Type: `Base_PowerController`.

**FVP_Base_Cortex_X1C.pctl.timer_reset**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_X1C.pctl.timer_reset.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_X1C.pctl.timer_reset.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_X1C.pctl.timer_reset.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

# 13.65 FVP_Base_Cortex-X2

List of instances in FVP_Base_Cortex-X2.

## FVP_Base_Cortex-X2 instances

**FVP_Base_Cortex_X2**

Base Platform Compute Subsystem for ARMCortexX2CT.
Type: `FVP_Base_Cortex_X2`.

**FVP_Base_Cortex_X2.bp**

Peripherals and address map for the Base Platform.
Type: `BasePlatformPeripherals`.

**FVP_Base_Cortex_X2.bp.Timer_0_1**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Cortex_X2.bp.Timer_0_1.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X2.bp.Timer_0_1.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X2.bp.Timer_0_1.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_X2.bp.Timer_0_1.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_X2.bp.Timer_2_3**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_X2.bp.Timer_2_3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X2.bp.Timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X2.bp.Timer_2_3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_X2.bp.Timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_X2.bp.ap_refclk**

ARM Generic Timer.
Type: `MemoryMappedGenericTimer`.

**FVP_Base_Cortex_X2.bp.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**FVP_Base_Cortex_X2.bp.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X2.bp.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X2.bp.clock300MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_X2.bp.clock32KHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X2.bp.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X2.bp.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X2.bp.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X2.bp.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X2.bp.dmc**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_X2.bp.dmc_phy**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_X2.bp.dummy_local_dap_rom**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_X2.bp.dummy_ram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_X2.bp.dummy_usb**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_X2.bp.exclusive_monitor**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_Base_Cortex_X2.bp.flash0**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_X2.bp.flash1**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_X2.bp.flashloader0**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_X2.bp.flashloader1**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_X2.bp.generic_watchdog**

ARM Generic Watchdog.
Type: `MemoryMappedGenericWatchdog`.

**FVP_Base_Cortex_X2.bp.hdlcd0**

ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).
Type: `PL370_HDLCD`.

**FVP_Base_Cortex_X2.bp.hdlcd0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_X2.bp.hdlcd0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_X2.bp.hdlcd0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_X2.bp.hdlcd0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_X2.bp.hdlcd0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_X2.bp.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_Base_Cortex_X2.bp.mmc**

Generic Multimedia Card.
Type: `MMC`.

**FVP_Base_Cortex_X2.bp.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_X2.bp.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_X2.bp.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_X2.bp.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_X2.bp.mpe**

Encrypt memory transactions for each encryption context with an independent key to prevent mismatch access.
Type: `PVMemoryProtectionEngine`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_X2.bp.nontrustedrom**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_X2.bp.nontrustedromloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_X2.bp.ns_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_X2.bp.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_X2.bp.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X2.bp.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_X2.bp.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_X2.bp.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_X2.bp.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_X2.bp.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_X2.bp.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X2.bp.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_X2.bp.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_X2.bp.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_X2.bp.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_X2.bp.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_X2.bp.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_X2.bp.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_X2.bp.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_X2.bp.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_X2.bp.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_X2.bp.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_X2.bp.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X2.bp.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_X2.bp.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_X2.bp.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_X2.bp.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_X2.bp.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

**FVP_Base_Cortex_X2.bp.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_X2.bp.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_X2.bp.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_X2.bp.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_X2.bp.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_Base_Cortex_X2.bp.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_X2.bp.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_X2.bp.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_X2.bp.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_X2.bp.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_X2.bp.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X2.bp.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_X2.bp.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X2.bp.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_Base_Cortex_X2.bp.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

**FVP_Base_Cortex_X2.bp.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_X2.bp.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_X2.bp.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_X2.bp.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_X2.bp.pl111_clcd_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_X2.bp.pl180_mci**

ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

**FVP_Base_Cortex_X2.bp.ps2keyboard**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_X2.bp.ps2keyboard.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_X2.bp.ps2keyboard.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_X2.bp.ps2keyboard.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_X2.bp.ps2keyboard.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_X2.bp.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

**FVP_Base_Cortex_X2.bp.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_X2.bp.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_X2.bp.ps2mouse.ps2_clocktimer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_X2.bp.ps2mouse.ps2_clocktimer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_X2.bp.psram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_X2.bp.refcounter**
Memory Mapped Counter Module for Generic Timers.
Type: `MemoryMappedCounterModule`.

**FVP_Base_Cortex_X2.bp.reset_or**
Or Gate.
Type: `OrGate`.

**FVP_Base_Cortex_X2.bp.rl_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_X2.bp.rt_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_X2.bp.s_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_X2.bp.secureDRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_X2.bp.secureSRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_X2.bp.secureflash**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_X2.bp.secureflashloader**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_X2.bp.smsc_91c111**

SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_Base_Cortex_X2.bp.sp805_wdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_X2.bp.sp805_wdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_X2.bp.sp805_wdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_X2.bp.sp805_wdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_X2.bp.sp805_wdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_X2.bp.sp810_sysctrl**

Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_Base_Cortex_X2.bp.sp810_sysctrl.clkdiv_clk0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X2.bp.sp810_sysctrl.clkdiv_clk1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X2.bp.sp810_sysctrl.clkdiv_clk2**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X2.bp.sp810_sysctrl.clkdiv_clk3**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X2.bp.sram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_X2.bp.terminal_0**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_X2.bp.terminal_1**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_X2.bp.terminal_2**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_X2.bp.terminal_3**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_X2.bp.trusted_key_storage**

Trusted Root-Key Storage unit.
Type: `RootKeyStorage`.

**FVP_Base_Cortex_X2.bp.trusted_nv_counter**

Trusted Non-Volatile Counter unit.
Type: `NonVolatileCounter`.

**FVP_Base_Cortex_X2.bp.trusted_rng**

Random Number Generator unit.
Type: `RandomNumberGenerator`.

**FVP_Base_Cortex_X2.bp.trusted_watchdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_X2.bp.trusted_watchdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_X2.bp.trusted_watchdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_X2.bp.trusted_watchdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_X2.bp.trusted_watchdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_X2.bp.tzc_400**

TrustZone Address Space Controller.
Type: `TZC_400`.

**FVP_Base_Cortex_X2.bp.ve_sysregs**

Type: `VE_SysRegs`.

**FVP_Base_Cortex_X2.bp.vedcc**

Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

**FVP_Base_Cortex_X2.bp.virtio_net**

VirtioNet device over MMIO transport.
Type: `VirtioNetMMIO`.

**FVP_Base_Cortex_X2.bp.virtio_net_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_X2.bp.virtio_rng**

VirtioEntropy device over MMIO transport.
Type: `VirtioEntropyMMIO`.

**FVP_Base_Cortex_X2.bp.virtioblockdevice**

VirtioBlock device over MMIO transport.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `VirtioBlockMMIO`.

**FVP_Base_Cortex_X2.bp.virtioblockdevice_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_X2.bp.virtiop9device**

virtio P9 server.
Type: `VirtioP9Device`.

**FVP_Base_Cortex_X2.bp.virtiop9device_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_X2.bp.vis**

Display window for VE using Visualisation library.
Type: `VEVisualisation`.

**FVP_Base_Cortex_X2.bp.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_Base_Cortex_X2.bp.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X2.bp.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_X2.bp.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_X2.bp.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_X2.bp.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_X2.bp.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X2.bp.vram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_X2.cci400**

Cache Coherent Interconnect for AXI4 ACE.
Type: `CCI400`.

**FVP_Base_Cortex_X2.clockdivider0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X2.cluster0**

ARM Cortex-X2 Cluster CT model.
Type: `Cluster_ARM_Cortex-X2`.

**FVP_Base_Cortex_X2.cluster0.DSU.shared_cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X2.cluster0.cpu0**

ARM Cortex-X2 CT model.
Type: `ARM_Cortex-X2`.

**FVP_Base_Cortex_X2.cluster0.cpu0.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_X2.cluster0.cpu0.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X2.cluster0.cpu0.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X2.cluster0.cpu0.l2cache**

PV Cache.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `PVCache`.

**`FVP_Base_Cortex_X2.cluster0.cpu1`**

ARM Cortex-X2 CT model.
Type: `ARM_Cortex-X2`.

**`FVP_Base_Cortex_X2.cluster0.cpu1.dtlb`**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**`FVP_Base_Cortex_X2.cluster0.cpu1.l1dcache`**

PV Cache.
Type: `PVCache`.

**`FVP_Base_Cortex_X2.cluster0.cpu1.l1icache`**

PV Cache.
Type: `PVCache`.

**`FVP_Base_Cortex_X2.cluster0.cpu1.l2cache`**

PV Cache.
Type: `PVCache`.

**`FVP_Base_Cortex_X2.cluster0.cpu10`**

ARM Cortex-X2 CT model.
Type: `ARM_Cortex-X2`.

**`FVP_Base_Cortex_X2.cluster0.cpu10.dtlb`**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**`FVP_Base_Cortex_X2.cluster0.cpu10.l1dcache`**

PV Cache.
Type: `PVCache`.

**`FVP_Base_Cortex_X2.cluster0.cpu10.l1icache`**

PV Cache.
Type: `PVCache`.

**`FVP_Base_Cortex_X2.cluster0.cpu10.l2cache`**

PV Cache.
Type: `PVCache`.

**`FVP_Base_Cortex_X2.cluster0.cpu11`**

ARM Cortex-X2 CT model.
Type: `ARM_Cortex-X2`.

**`FVP_Base_Cortex_X2.cluster0.cpu11.dtlb`**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**`FVP_Base_Cortex_X2.cluster0.cpu11.l1dcache`**

PV Cache.
Type: `PVCache`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_X2.cluster0.cpu11.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X2.cluster0.cpu11.l2cache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X2.cluster0.cpu2**
ARM Cortex-X2 CT model.
Type: `ARM_Cortex-X2`.

**FVP_Base_Cortex_X2.cluster0.cpu2.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_X2.cluster0.cpu2.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X2.cluster0.cpu2.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X2.cluster0.cpu2.l2cache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X2.cluster0.cpu3**
ARM Cortex-X2 CT model.
Type: `ARM_Cortex-X2`.

**FVP_Base_Cortex_X2.cluster0.cpu3.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_X2.cluster0.cpu3.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X2.cluster0.cpu3.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X2.cluster0.cpu3.l2cache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X2.cluster0.cpu4**
ARM Cortex-X2 CT model.
Type: `ARM_Cortex-X2`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_X2.cluster0.cpu4.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_X2.cluster0.cpu4.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X2.cluster0.cpu4.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X2.cluster0.cpu4.l2cache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X2.cluster0.cpu5**
ARM Cortex-X2 CT model.
Type: `ARM_Cortex-X2`.

**FVP_Base_Cortex_X2.cluster0.cpu5.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_X2.cluster0.cpu5.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X2.cluster0.cpu5.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X2.cluster0.cpu5.l2cache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X2.cluster0.cpu6**
ARM Cortex-X2 CT model.
Type: `ARM_Cortex-X2`.

**FVP_Base_Cortex_X2.cluster0.cpu6.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_X2.cluster0.cpu6.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X2.cluster0.cpu6.l1icache**
PV Cache.
Type: `PVCache`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_X2.cluster0.cpu6.l2cache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X2.cluster0.cpu7**
ARM Cortex-X2 CT model.
Type: `ARM_Cortex-X2`.

**FVP_Base_Cortex_X2.cluster0.cpu7.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_X2.cluster0.cpu7.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X2.cluster0.cpu7.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X2.cluster0.cpu7.l2cache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X2.cluster0.cpu8**
ARM Cortex-X2 CT model.
Type: `ARM_Cortex-X2`.

**FVP_Base_Cortex_X2.cluster0.cpu8.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_X2.cluster0.cpu8.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X2.cluster0.cpu8.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X2.cluster0.cpu8.l2cache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X2.cluster0.cpu9**
ARM Cortex-X2 CT model.
Type: `ARM_Cortex-X2`.

**FVP_Base_Cortex_X2.cluster0.cpu9.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_X2.cluster0.cpu9.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X2.cluster0.cpu9.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X2.cluster0.cpu9.l2cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X2.cluster0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_X2.dapmemlogger**

Bus Logger.
Type: `PVBusLogger`.

**FVP_Base_Cortex_X2.elfloader**

ELF loader component.
Type: `ElfLoader`.

**FVP_Base_Cortex_X2.gic_distributor**

GIC Interrupt Redistribution Infrastucture component.
Type: `GIC_IRI`.

**FVP_Base_Cortex_X2.pctl**

Base Platforms Power Controller.
Type: `Base_PowerController`.

**FVP_Base_Cortex_X2.pctl.timer_reset**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_X2.pctl.timer_reset.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_X2.pctl.timer_reset.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `SchedulerThread`.

**`FVP_Base_Cortex_X2.pctl.timer_reset.timer.thread_event`**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

## 13.66 FVP_Base_Cortex-X3

List of instances in FVP_Base_Cortex-X3.

### FVP_Base_Cortex-X3 instances

**`FVP_Base_Cortex_X3`**

Base Platform Compute Subsystem for ARMCortexX3CT.
Type: `FVP_Base_Cortex_X3`.

**`FVP_Base_Cortex_X3.bp`**

Peripherals and address map for the Base Platform.
Type: `BasePlatformPeripherals`.

**`FVP_Base_Cortex_X3.bp.Timer_0_1`**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**`FVP_Base_Cortex_X3.bp.Timer_0_1.clk_div0`**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**`FVP_Base_Cortex_X3.bp.Timer_0_1.clk_div1`**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**`FVP_Base_Cortex_X3.bp.Timer_0_1.counter0`**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**`FVP_Base_Cortex_X3.bp.Timer_0_1.counter1`**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**`FVP_Base_Cortex_X3.bp.Timer_2_3`**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_X3.bp.Timer_2_3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X3.bp.Timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X3.bp.Timer_2_3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_X3.bp.Timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_X3.bp.ap_refclk**

ARM Generic Timer.
Type: `MemoryMappedGenericTimer`.

**FVP_Base_Cortex_X3.bp.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**FVP_Base_Cortex_X3.bp.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X3.bp.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X3.bp.clock300MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_X3.bp.clock32KHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X3.bp.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X3.bp.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X3.bp.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X3.bp.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X3.bp.dmc**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_X3.bp.dmc_phy**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_X3.bp.dummy_local_dap_rom**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_X3.bp.dummy_ram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_X3.bp.dummy_usb**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_X3.bp.exclusive_monitor**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_Base_Cortex_X3.bp.flash0**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_X3.bp.flash1**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_X3.bp.flashloader0**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_X3.bp.flashloader1**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_X3.bp.generic_watchdog**

ARM Generic Watchdog.
Type: `MemoryMappedGenericWatchdog`.

**FVP_Base_Cortex_X3.bp.hdlcd0**

ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).
Type: `PL370_HDLCD`.

**FVP_Base_Cortex_X3.bp.hdlcd0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_X3.bp.hdlcd0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_X3.bp.hdlcd0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_X3.bp.hdlcd0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_X3.bp.hdlcd0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_X3.bp.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_Base_Cortex_X3.bp.mmc**

Generic Multimedia Card.
Type: `MMC`.

**FVP_Base_Cortex_X3.bp.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_X3.bp.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_X3.bp.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_X3.bp.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_X3.bp.mpe**

Encrypt memory transactions for each encryption context with an independent key to prevent mismatch access.
Type: `PVMemoryProtectionEngine`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_X3.bp.nontrustedrom**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_X3.bp.nontrustedromloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_X3.bp.ns_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_X3.bp.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_X3.bp.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X3.bp.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_X3.bp.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_X3.bp.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_X3.bp.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_X3.bp.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_X3.bp.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X3.bp.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_X3.bp.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_X3.bp.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_X3.bp.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_X3.bp.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_X3.bp.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_X3.bp.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_X3.bp.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_X3.bp.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_X3.bp.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_X3.bp.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_X3.bp.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X3.bp.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_X3.bp.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_X3.bp.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_X3.bp.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_X3.bp.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

**FVP_Base_Cortex_X3.bp.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_X3.bp.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_X3.bp.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_X3.bp.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_X3.bp.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_Base_Cortex_X3.bp.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_X3.bp.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_X3.bp.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_X3.bp.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_X3.bp.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_X3.bp.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X3.bp.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_X3.bp.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X3.bp.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_Base_Cortex_X3.bp.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

**FVP_Base_Cortex_X3.bp.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_X3.bp.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_X3.bp.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_X3.bp.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_X3.bp.pl111_clcd_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_X3.bp.pl180_mci**

ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

**FVP_Base_Cortex_X3.bp.ps2keyboard**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_X3.bp.ps2keyboard.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_X3.bp.ps2keyboard.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_X3.bp.ps2keyboard.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_X3.bp.ps2keyboard.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_X3.bp.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

**FVP_Base_Cortex_X3.bp.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_X3.bp.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_X3.bp.ps2mouse.ps2_clocktimer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_X3.bp.ps2mouse.ps2_clocktimer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_X3.bp.psram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_X3.bp.refcounter**
Memory Mapped Counter Module for Generic Timers.
Type: `MemoryMappedCounterModule`.

**FVP_Base_Cortex_X3.bp.reset_or**
Or Gate.
Type: `OrGate`.

**FVP_Base_Cortex_X3.bp.rl_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_X3.bp.rt_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_X3.bp.s_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_X3.bp.secureDRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_X3.bp.secureSRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_X3.bp.secureflash**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_X3.bp.secureflashloader**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_X3.bp.smsc_91c111**

SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_Base_Cortex_X3.bp.sp805_wdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_X3.bp.sp805_wdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_X3.bp.sp805_wdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_X3.bp.sp805_wdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_X3.bp.sp805_wdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_X3.bp.sp810_sysctrl**

Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_Base_Cortex_X3.bp.sp810_sysctrl.clkdiv_clk0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X3.bp.sp810_sysctrl.clkdiv_clk1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X3.bp.sp810_sysctrl.clkdiv_clk2**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X3.bp.sp810_sysctrl.clkdiv_clk3**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X3.bp.sram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_X3.bp.terminal_0**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_X3.bp.terminal_1**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_X3.bp.terminal_2**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_X3.bp.terminal_3**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_X3.bp.trusted_key_storage**

Trusted Root-Key Storage unit.
Type: `RootKeyStorage`.

**FVP_Base_Cortex_X3.bp.trusted_nv_counter**

Trusted Non-Volatile Counter unit.
Type: `NonVolatileCounter`.

**FVP_Base_Cortex_X3.bp.trusted_rng**

Random Number Generator unit.
Type: `RandomNumberGenerator`.

**FVP_Base_Cortex_X3.bp.trusted_watchdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_X3.bp.trusted_watchdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_X3.bp.trusted_watchdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_X3.bp.trusted_watchdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_X3.bp.trusted_watchdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_X3.bp.tzc_400**

TrustZone Address Space Controller.
Type: `TZC_400`.

**FVP_Base_Cortex_X3.bp.ve_sysregs**

Type: `VE_SysRegs`.

**FVP_Base_Cortex_X3.bp.vedcc**

Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

**FVP_Base_Cortex_X3.bp.virtio_net**

VirtioNet device over MMIO transport.
Type: `VirtioNetMMIO`.

**FVP_Base_Cortex_X3.bp.virtio_net_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_X3.bp.virtio_rng**

VirtioEntropy device over MMIO transport.
Type: `VirtioEntropyMMIO`.

**FVP_Base_Cortex_X3.bp.virtioblockdevice**

VirtioBlock device over MMIO transport.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `VirtioBlockMMIO`.

**FVP_Base_Cortex_X3.bp.virtioblockdevice_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_X3.bp.virtiop9device**

virtio P9 server.
Type: `VirtioP9Device`.

**FVP_Base_Cortex_X3.bp.virtiop9device_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_X3.bp.vis**

Display window for VE using Visualisation library.
Type: `VEVisualisation`.

**FVP_Base_Cortex_X3.bp.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_Base_Cortex_X3.bp.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X3.bp.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_X3.bp.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_X3.bp.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_X3.bp.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_X3.bp.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X3.bp.vram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_X3.cci400**

Cache Coherent Interconnect for AXI4 ACE.
Type: `CCI400`.

**FVP_Base_Cortex_X3.clockdivider0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X3.cluster0**

ARM Cortex-X3 Cluster CT model.
Type: `Cluster_ARM_Cortex-X3`.

**FVP_Base_Cortex_X3.cluster0.DSU.shared_cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X3.cluster0.cpu0**

ARM Cortex-X3 CT model.
Type: `ARM_Cortex-X3`.

**FVP_Base_Cortex_X3.cluster0.cpu0.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_X3.cluster0.cpu0.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X3.cluster0.cpu0.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X3.cluster0.cpu0.l2cache**

PV Cache.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `PVCache`.

**`FVP_Base_Cortex_X3.cluster0.cpu1`**

ARM Cortex-X3 CT model.
Type: `ARM_Cortex-X3`.

**`FVP_Base_Cortex_X3.cluster0.cpu1.dtlb`**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**`FVP_Base_Cortex_X3.cluster0.cpu1.l1dcache`**

PV Cache.
Type: `PVCache`.

**`FVP_Base_Cortex_X3.cluster0.cpu1.l1icache`**

PV Cache.
Type: `PVCache`.

**`FVP_Base_Cortex_X3.cluster0.cpu1.l2cache`**

PV Cache.
Type: `PVCache`.

**`FVP_Base_Cortex_X3.cluster0.cpu10`**

ARM Cortex-X3 CT model.
Type: `ARM_Cortex-X3`.

**`FVP_Base_Cortex_X3.cluster0.cpu10.dtlb`**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**`FVP_Base_Cortex_X3.cluster0.cpu10.l1dcache`**

PV Cache.
Type: `PVCache`.

**`FVP_Base_Cortex_X3.cluster0.cpu10.l1icache`**

PV Cache.
Type: `PVCache`.

**`FVP_Base_Cortex_X3.cluster0.cpu10.l2cache`**

PV Cache.
Type: `PVCache`.

**`FVP_Base_Cortex_X3.cluster0.cpu11`**

ARM Cortex-X3 CT model.
Type: `ARM_Cortex-X3`.

**`FVP_Base_Cortex_X3.cluster0.cpu11.dtlb`**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**`FVP_Base_Cortex_X3.cluster0.cpu11.l1dcache`**

PV Cache.
Type: `PVCache`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_X3.cluster0.cpu11.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X3.cluster0.cpu11.l2cache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X3.cluster0.cpu2**
ARM Cortex-X3 CT model.
Type: `ARM_Cortex-X3`.

**FVP_Base_Cortex_X3.cluster0.cpu2.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_X3.cluster0.cpu2.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X3.cluster0.cpu2.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X3.cluster0.cpu2.l2cache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X3.cluster0.cpu3**
ARM Cortex-X3 CT model.
Type: `ARM_Cortex-X3`.

**FVP_Base_Cortex_X3.cluster0.cpu3.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_X3.cluster0.cpu3.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X3.cluster0.cpu3.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X3.cluster0.cpu3.l2cache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X3.cluster0.cpu4**
ARM Cortex-X3 CT model.
Type: `ARM_Cortex-X3`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_X3.cluster0.cpu4.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_X3.cluster0.cpu4.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X3.cluster0.cpu4.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X3.cluster0.cpu4.l2cache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X3.cluster0.cpu5**
ARM Cortex-X3 CT model.
Type: `ARM_Cortex-X3`.

**FVP_Base_Cortex_X3.cluster0.cpu5.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_X3.cluster0.cpu5.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X3.cluster0.cpu5.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X3.cluster0.cpu5.l2cache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X3.cluster0.cpu6**
ARM Cortex-X3 CT model.
Type: `ARM_Cortex-X3`.

**FVP_Base_Cortex_X3.cluster0.cpu6.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_X3.cluster0.cpu6.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X3.cluster0.cpu6.l1icache**
PV Cache.
Type: `PVCache`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_X3.cluster0.cpu6.l2cache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X3.cluster0.cpu7**
ARM Cortex-X3 CT model.
Type: `ARM_Cortex-X3`.

**FVP_Base_Cortex_X3.cluster0.cpu7.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_X3.cluster0.cpu7.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X3.cluster0.cpu7.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X3.cluster0.cpu7.l2cache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X3.cluster0.cpu8**
ARM Cortex-X3 CT model.
Type: `ARM_Cortex-X3`.

**FVP_Base_Cortex_X3.cluster0.cpu8.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_X3.cluster0.cpu8.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X3.cluster0.cpu8.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X3.cluster0.cpu8.l2cache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X3.cluster0.cpu9**
ARM Cortex-X3 CT model.
Type: `ARM_Cortex-X3`.

**FVP_Base_Cortex_X3.cluster0.cpu9.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_X3.cluster0.cpu9.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X3.cluster0.cpu9.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X3.cluster0.cpu9.l2cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X3.cluster0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_X3.dapmemlogger**

Bus Logger.
Type: `PVBusLogger`.

**FVP_Base_Cortex_X3.elfloader**

ELF loader component.
Type: `ElfLoader`.

**FVP_Base_Cortex_X3.gic_distributor**

GIC Interrupt Redistribution Infrastucture component.
Type: `GIC_IRI`.

**FVP_Base_Cortex_X3.pctl**

Base Platforms Power Controller.
Type: `Base_PowerController`.

**FVP_Base_Cortex_X3.pctl.timer_reset**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_X3.pctl.timer_reset.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_X3.pctl.timer_reset.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `SchedulerThread`.

**`FVP_Base_Cortex_X3.pctl.timer_reset.timer.thread_event`**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

## 13.67 FVP_Base_Cortex-X4

List of instances in FVP_Base_Cortex-X4.

### FVP_Base_Cortex-X4 instances

**`FVP_Base_Cortex_X4`**

Base Platform Compute Subsystem for ARMCortexX4CT.
Type: `FVP_Base_Cortex_X4`.

**`FVP_Base_Cortex_X4.bp`**

Peripherals and address map for the Base Platform.
Type: `BasePlatformPeripherals`.

**`FVP_Base_Cortex_X4.bp.Timer_0_1`**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**`FVP_Base_Cortex_X4.bp.Timer_0_1.clk_div0`**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**`FVP_Base_Cortex_X4.bp.Timer_0_1.clk_div1`**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**`FVP_Base_Cortex_X4.bp.Timer_0_1.counter0`**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**`FVP_Base_Cortex_X4.bp.Timer_0_1.counter1`**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**`FVP_Base_Cortex_X4.bp.Timer_2_3`**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_X4.bp.Timer_2_3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X4.bp.Timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X4.bp.Timer_2_3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_X4.bp.Timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Cortex_X4.bp.ap_refclk**

ARM Generic Timer.
Type: `MemoryMappedGenericTimer`.

**FVP_Base_Cortex_X4.bp.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**FVP_Base_Cortex_X4.bp.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X4.bp.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X4.bp.clock300MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_X4.bp.clock32KHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X4.bp.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X4.bp.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X4.bp.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X4.bp.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X4.bp.dmc**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_X4.bp.dmc_phy**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_X4.bp.dummy_local_dap_rom**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_X4.bp.dummy_ram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_X4.bp.dummy_usb**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_X4.bp.exclusive_monitor**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_Base_Cortex_X4.bp.flash0**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_X4.bp.flash1**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_X4.bp.flashloader0**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_X4.bp.flashloader1**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_X4.bp.generic_watchdog**

ARM Generic Watchdog.
Type: `MemoryMappedGenericWatchdog`.

**FVP_Base_Cortex_X4.bp.hdlcd0**

ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).
Type: `PL370_HDLCD`.

**FVP_Base_Cortex_X4.bp.hdlcd0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_X4.bp.hdlcd0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_X4.bp.hdlcd0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_X4.bp.hdlcd0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_X4.bp.hdlcd0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_X4.bp.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_Base_Cortex_X4.bp.mmc**

Generic Multimedia Card.
Type: `MMC`.

**FVP_Base_Cortex_X4.bp.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_X4.bp.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_X4.bp.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_X4.bp.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_X4.bp.mpe**

Encrypt memory transactions for each encryption context with an independent key to prevent mismatch access.
Type: `PVMemoryProtectionEngine`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_X4.bp.nontrustedrom**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_X4.bp.nontrustedromloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Cortex_X4.bp.ns_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_X4.bp.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_X4.bp.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X4.bp.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_X4.bp.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_X4.bp.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_X4.bp.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_X4.bp.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_X4.bp.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X4.bp.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_X4.bp.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_X4.bp.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_X4.bp.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_X4.bp.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_X4.bp.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_X4.bp.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_X4.bp.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_X4.bp.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_X4.bp.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_X4.bp.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Cortex_X4.bp.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X4.bp.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_X4.bp.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_X4.bp.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_X4.bp.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_X4.bp.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

**FVP_Base_Cortex_X4.bp.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_X4.bp.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_X4.bp.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_X4.bp.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_X4.bp.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_Base_Cortex_X4.bp.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_X4.bp.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_X4.bp.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_X4.bp.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_X4.bp.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_X4.bp.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X4.bp.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Cortex_X4.bp.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X4.bp.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_Base_Cortex_X4.bp.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

**FVP_Base_Cortex_X4.bp.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_X4.bp.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_X4.bp.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_X4.bp.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_X4.bp.pl111_clcd_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_X4.bp.pl180_mci**

ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

**FVP_Base_Cortex_X4.bp.ps2keyboard**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_X4.bp.ps2keyboard.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_X4.bp.ps2keyboard.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_X4.bp.ps2keyboard.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_X4.bp.ps2keyboard.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_X4.bp.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

**FVP_Base_Cortex_X4.bp.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_X4.bp.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_X4.bp.ps2mouse.ps2_clocktimer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_X4.bp.ps2mouse.ps2_clocktimer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_X4.bp.psram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_X4.bp.refcounter**
Memory Mapped Counter Module for Generic Timers.
Type: `MemoryMappedCounterModule`.

**FVP_Base_Cortex_X4.bp.reset_or**
Or Gate.
Type: `OrGate`.

**FVP_Base_Cortex_X4.bp.rl_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_X4.bp.rt_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_X4.bp.s_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_X4.bp.secureDRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_X4.bp.secureSRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_X4.bp.secureflash**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Cortex_X4.bp.secureflashloader**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_X4.bp.smsc_91c111**

SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_Base_Cortex_X4.bp.sp805_wdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Cortex_X4.bp.sp805_wdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_X4.bp.sp805_wdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_X4.bp.sp805_wdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_X4.bp.sp805_wdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_X4.bp.sp810_sysctrl**

Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_Base_Cortex_X4.bp.sp810_sysctrl.clkdiv_clk0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X4.bp.sp810_sysctrl.clkdiv_clk1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X4.bp.sp810_sysctrl.clkdiv_clk2**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X4.bp.sp810_sysctrl.clkdiv_clk3**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X4.bp.sram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_X4.bp.terminal_0**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_X4.bp.terminal_1**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_X4.bp.terminal_2**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_X4.bp.terminal_3**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Cortex_X4.bp.trusted_key_storage**

Trusted Root-Key Storage unit.
Type: `RootKeyStorage`.

**FVP_Base_Cortex_X4.bp.trusted_nv_counter**

Trusted Non-Volatile Counter unit.
Type: `NonVolatileCounter`.

**FVP_Base_Cortex_X4.bp.trusted_rng**

Random Number Generator unit.
Type: `RandomNumberGenerator`.

**FVP_Base_Cortex_X4.bp.trusted_watchdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_X4.bp.trusted_watchdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_X4.bp.trusted_watchdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_X4.bp.trusted_watchdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_X4.bp.trusted_watchdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_X4.bp.tzc_400**

TrustZone Address Space Controller.
Type: `TZC_400`.

**FVP_Base_Cortex_X4.bp.ve_sysregs**

Type: `VE_SysRegs`.

**FVP_Base_Cortex_X4.bp.vedcc**

Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

**FVP_Base_Cortex_X4.bp.virtio_net**

VirtioNet device over MMIO transport.
Type: `VirtioNetMMIO`.

**FVP_Base_Cortex_X4.bp.virtio_net_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_X4.bp.virtio_rng**

VirtioEntropy device over MMIO transport.
Type: `VirtioEntropyMMIO`.

**FVP_Base_Cortex_X4.bp.virtioblockdevice**

VirtioBlock device over MMIO transport.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `VirtioBlockMMIO`.

**FVP_Base_Cortex_X4.bp.virtioblockdevice_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_X4.bp.virtiop9device**

virtio P9 server.
Type: `VirtioP9Device`.

**FVP_Base_Cortex_X4.bp.virtiop9device_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_X4.bp.vis**

Display window for VE using Visualisation library.
Type: `VEVisualisation`.

**FVP_Base_Cortex_X4.bp.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_Base_Cortex_X4.bp.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X4.bp.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_X4.bp.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_X4.bp.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_X4.bp.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Cortex_X4.bp.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X4.bp.vram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Cortex_X4.cci400**

Cache Coherent Interconnect for AXI4 ACE.
Type: `CCI400`.

**FVP_Base_Cortex_X4.clockdivider0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Cortex_X4.cluster0**

ARM Cortex-X4 Cluster CT model.
Type: `Cluster_ARM_Cortex-X4`.

**FVP_Base_Cortex_X4.cluster0.DSU.shared_cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X4.cluster0.cpu0**

ARM Cortex-X4 CT model.
Type: `ARM_Cortex-X4`.

**FVP_Base_Cortex_X4.cluster0.cpu0.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_X4.cluster0.cpu0.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X4.cluster0.cpu0.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X4.cluster0.cpu0.l2cache**

PV Cache.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `PVCache`.

**`FVP_Base_Cortex_X4.cluster0.cpu1`**

ARM Cortex-X4 CT model.
Type: `ARM_Cortex-X4`.

**`FVP_Base_Cortex_X4.cluster0.cpu1.dtlb`**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**`FVP_Base_Cortex_X4.cluster0.cpu1.l1dcache`**

PV Cache.
Type: `PVCache`.

**`FVP_Base_Cortex_X4.cluster0.cpu1.l1icache`**

PV Cache.
Type: `PVCache`.

**`FVP_Base_Cortex_X4.cluster0.cpu1.l2cache`**

PV Cache.
Type: `PVCache`.

**`FVP_Base_Cortex_X4.cluster0.cpu10`**

ARM Cortex-X4 CT model.
Type: `ARM_Cortex-X4`.

**`FVP_Base_Cortex_X4.cluster0.cpu10.dtlb`**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**`FVP_Base_Cortex_X4.cluster0.cpu10.l1dcache`**

PV Cache.
Type: `PVCache`.

**`FVP_Base_Cortex_X4.cluster0.cpu10.l1icache`**

PV Cache.
Type: `PVCache`.

**`FVP_Base_Cortex_X4.cluster0.cpu10.l2cache`**

PV Cache.
Type: `PVCache`.

**`FVP_Base_Cortex_X4.cluster0.cpu11`**

ARM Cortex-X4 CT model.
Type: `ARM_Cortex-X4`.

**`FVP_Base_Cortex_X4.cluster0.cpu11.dtlb`**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**`FVP_Base_Cortex_X4.cluster0.cpu11.l1dcache`**

PV Cache.
Type: `PVCache`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_X4.cluster0.cpu11.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X4.cluster0.cpu11.l2cache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X4.cluster0.cpu12**
ARM Cortex-X4 CT model.
Type: `ARM_Cortex-X4`.

**FVP_Base_Cortex_X4.cluster0.cpu12.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_X4.cluster0.cpu12.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X4.cluster0.cpu12.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X4.cluster0.cpu12.l2cache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X4.cluster0.cpu13**
ARM Cortex-X4 CT model.
Type: `ARM_Cortex-X4`.

**FVP_Base_Cortex_X4.cluster0.cpu13.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_X4.cluster0.cpu13.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X4.cluster0.cpu13.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X4.cluster0.cpu13.l2cache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X4.cluster0.cpu2**
ARM Cortex-X4 CT model.
Type: `ARM_Cortex-X4`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_X4.cluster0.cpu2.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_X4.cluster0.cpu2.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X4.cluster0.cpu2.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X4.cluster0.cpu2.l2cache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X4.cluster0.cpu3**
ARM Cortex-X4 CT model.
Type: `ARM_Cortex-X4`.

**FVP_Base_Cortex_X4.cluster0.cpu3.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_X4.cluster0.cpu3.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X4.cluster0.cpu3.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X4.cluster0.cpu3.l2cache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X4.cluster0.cpu4**
ARM Cortex-X4 CT model.
Type: `ARM_Cortex-X4`.

**FVP_Base_Cortex_X4.cluster0.cpu4.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_X4.cluster0.cpu4.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X4.cluster0.cpu4.l1icache**
PV Cache.
Type: `PVCache`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_X4.cluster0.cpu4.l2cache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X4.cluster0.cpu5**
ARM Cortex-X4 CT model.
Type: `ARM_Cortex-X4`.

**FVP_Base_Cortex_X4.cluster0.cpu5.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_X4.cluster0.cpu5.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X4.cluster0.cpu5.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X4.cluster0.cpu5.l2cache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X4.cluster0.cpu6**
ARM Cortex-X4 CT model.
Type: `ARM_Cortex-X4`.

**FVP_Base_Cortex_X4.cluster0.cpu6.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_X4.cluster0.cpu6.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X4.cluster0.cpu6.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X4.cluster0.cpu6.l2cache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X4.cluster0.cpu7**
ARM Cortex-X4 CT model.
Type: `ARM_Cortex-X4`.

**FVP_Base_Cortex_X4.cluster0.cpu7.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_X4.cluster0.cpu7.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X4.cluster0.cpu7.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X4.cluster0.cpu7.l2cache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X4.cluster0.cpu8**
ARM Cortex-X4 CT model.
Type: `ARM_Cortex-X4`.

**FVP_Base_Cortex_X4.cluster0.cpu8.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_X4.cluster0.cpu8.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X4.cluster0.cpu8.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X4.cluster0.cpu8.l2cache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X4.cluster0.cpu9**
ARM Cortex-X4 CT model.
Type: `ARM_Cortex-X4`.

**FVP_Base_Cortex_X4.cluster0.cpu9.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Cortex_X4.cluster0.cpu9.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X4.cluster0.cpu9.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_Cortex_X4.cluster0.cpu9.l2cache**
PV Cache.
Type: `PVCache`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Cortex_X4.cluster0_labeller**

Type: `Labeller`.

**FVP_Base_Cortex_X4.dapmemlogger**

Bus Logger.
Type: `PVBusLogger`.

**FVP_Base_Cortex_X4.elfloader**

ELF loader component.
Type: `ElfLoader`.

**FVP_Base_Cortex_X4.gic_distributor**

GIC Interrupt Redistribution Infrastucture component.
Type: `GIC_IRI`.

**FVP_Base_Cortex_X4.pctl**

Base Platforms Power Controller.
Type: `Base_PowerController`.

**FVP_Base_Cortex_X4.pctl.timer_reset**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Cortex_X4.pctl.timer_reset.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Cortex_X4.pctl.timer_reset.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Cortex_X4.pctl.timer_reset.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

# 13.68 FVP_Base_Neoverse-E1

List of instances in FVP_Base_Neoverse-E1.

## FVP_Base_Neoverse-E1 instances

**FVP_Base_Neoverse_E1**

Base Platform Compute Subsystem for ARMNeoverseE1CT.
Type: `FVP_Base_Neoverse_E1`.

**FVP_Base_Neoverse_E1.bp**

Peripherals and address map for the Base Platform.
Type: `BasePlatformPeripherals`.

**FVP_Base_Neoverse_E1.bp.Timer_0_1**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Neoverse_E1.bp.Timer_0_1.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_E1.bp.Timer_0_1.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_E1.bp.Timer_0_1.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Neoverse_E1.bp.Timer_0_1.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Neoverse_E1.bp.Timer_2_3**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Neoverse_E1.bp.Timer_2_3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Neoverse_E1.bp.Timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_E1.bp.Timer_2_3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Neoverse_E1.bp.Timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Neoverse_E1.bp.ap_refclk**

ARM Generic Timer.
Type: `MemoryMappedGenericTimer`.

**FVP_Base_Neoverse_E1.bp.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**FVP_Base_Neoverse_E1.bp.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_E1.bp.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_E1.bp.clock300MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_E1.bp.clock32KHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Neoverse_E1.bp.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_E1.bp.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_E1.bp.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_E1.bp.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_E1.bp.dmc**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Neoverse_E1.bp.dmc_phy**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Neoverse_E1.bp.dummy_local_dap_rom**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Neoverse_E1.bp.dummy_ram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Neoverse_E1.bp.dummy_usb**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Neoverse_E1.bp.exclusive_monitor**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Neoverse_E1.bp.flash0**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Neoverse_E1.bp.flash1**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Neoverse_E1.bp.flashloader0**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Neoverse_E1.bp.flashloader1**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Neoverse_E1.bp.generic_watchdog**

ARM Generic Watchdog.
Type: `MemoryMappedGenericWatchdog`.

**FVP_Base_Neoverse_E1.bp.hdlcd0**

ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).
Type: `PL370_HDLCD`.

**FVP_Base_Neoverse_E1.bp.hdlcd0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Neoverse_E1.bp.hdlcd0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Neoverse_E1.bp.hdlcd0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Neoverse_E1.bp.hdlcd0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Neoverse_E1.bp.hdlcd0_labeller**

Type: `Labeller`.

**FVP_Base_Neoverse_E1.bp.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_Base_Neoverse_E1.bp.mmc**

Generic Multimedia Card.
Type: `MMC`.

**FVP_Base_Neoverse_E1.bp.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Neoverse_E1.bp.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Neoverse_E1.bp.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Neoverse_E1.bp.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Neoverse_E1.bp.mpe**

Encrypt memory transactions for each encryption context with an independent key to prevent mismatch access.
Type: `PVMemoryProtectionEngine`.

**FVP_Base_Neoverse_E1.bp.nontrustedrom**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Neoverse_E1.bp.nontrustedromloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Neoverse_E1.bp.ns_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Neoverse_E1.bp.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Neoverse_E1.bp.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_E1.bp.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Neoverse_E1.bp.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Neoverse_E1.bp.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Neoverse_E1.bp.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Neoverse_E1.bp.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Neoverse_E1.bp.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_E1.bp.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Neoverse_E1.bp.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Neoverse_E1.bp.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Neoverse_E1.bp.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Neoverse_E1.bp.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Neoverse_E1.bp.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_E1.bp.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ClockTimerThread`.

**FVP_Base_Neoverse_E1.bp.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Neoverse_E1.bp.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Neoverse_E1.bp.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Neoverse_E1.bp.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Neoverse_E1.bp.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_E1.bp.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Neoverse_E1.bp.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Neoverse_E1.bp.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Neoverse_E1.bp.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Neoverse_E1.bp.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

**FVP_Base_Neoverse_E1.bp.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Neoverse_E1.bp.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Neoverse_E1.bp.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Neoverse_E1.bp.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Neoverse_E1.bp.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_Base_Neoverse_E1.bp.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Neoverse_E1.bp.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Neoverse_E1.bp.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Neoverse_E1.bp.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Neoverse_E1.bp.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Neoverse_E1.bp.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_E1.bp.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Neoverse_E1.bp.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_E1.bp.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_Base_Neoverse_E1.bp.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Neoverse_E1.bp.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Neoverse_E1.bp.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Neoverse_E1.bp.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Neoverse_E1.bp.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Neoverse_E1.bp.pl111_clcd_labeller**

Type: `Labeller`.

**FVP_Base_Neoverse_E1.bp.pl180_mci**

ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

**FVP_Base_Neoverse_E1.bp.ps2keyboard**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

**FVP_Base_Neoverse_E1.bp.ps2keyboard.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Neoverse_E1.bp.ps2keyboard.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Neoverse_E1.bp.ps2keyboard.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Neoverse_E1.bp.ps2keyboard.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Neoverse_E1.bp.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

**FVP_Base_Neoverse_E1.bp.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Neoverse_E1.bp.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Neoverse_E1.bp.ps2mouse.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Neoverse_E1.bp.ps2mouse.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Neoverse_E1.bp.psram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Neoverse_E1.bp.refcounter**
Memory Mapped Counter Module for Generic Timers.
Type: `MemoryMappedCounterModule`.

**FVP_Base_Neoverse_E1.bp.reset_or**
Or Gate.
Type: `OrGate`.

**FVP_Base_Neoverse_E1.bp.rl_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Neoverse_E1.bp.rt_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Neoverse_E1.bp.s_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Neoverse_E1.bp.secureDRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Neoverse_E1.bp.secureSRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Neoverse_E1.bp.secureflash**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Neoverse_E1.bp.secureflashloader**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Neoverse_E1.bp.smsc_91c111**
SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_Base_Neoverse_E1.bp.sp805_wdog**
ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Neoverse_E1.bp.sp805_wdog.clocktimer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Neoverse_E1.bp.sp805_wdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Neoverse_E1.bp.sp805_wdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Neoverse_E1.bp.sp805_wdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Neoverse_E1.bp.sp810_sysctrl**

Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_Base_Neoverse_E1.bp.sp810_sysctrl.clkdiv_clk0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_E1.bp.sp810_sysctrl.clkdiv_clk1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_E1.bp.sp810_sysctrl.clkdiv_clk2**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_E1.bp.sp810_sysctrl.clkdiv_clk3**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_E1.bp.sram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Neoverse_E1.bp.terminal_0**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Neoverse_E1.bp.terminal_1**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Neoverse_E1.bp.terminal_2**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Neoverse_E1.bp.terminal_3**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Neoverse_E1.bp.trusted_key_storage**
Trusted Root-Key Storage unit.
Type: `RootKeyStorage`.

**FVP_Base_Neoverse_E1.bp.trusted_nv_counter**
Trusted Non-Volatile Counter unit.
Type: `NonVolatileCounter`.

**FVP_Base_Neoverse_E1.bp.trusted_rng**
Random Number Generator unit.
Type: `RandomNumberGenerator`.

**FVP_Base_Neoverse_E1.bp.trusted_watchdog**
ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Neoverse_E1.bp.trusted_watchdog.clocktimer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Neoverse_E1.bp.trusted_watchdog.clocktimer.timer**
A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Neoverse_E1.bp.trusted_watchdog.clocktimer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Neoverse_E1.bp.trusted_watchdog.clocktimer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Neoverse_E1.bp.tzc_400**
TrustZone Address Space Controller.
Type: `TZC_400`.

**FVP_Base_Neoverse_E1.bp.ve_sysregs**
Type: `VE_SysRegs`.

**FVP_Base_Neoverse_E1.bp.vedcc**
Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

**FVP_Base_Neoverse_E1.bp.virtio_net**
VirtioNet device over MMIO transport.
Type: `VirtioNetMMIO`.

**FVP_Base_Neoverse_E1.bp.virtio_net_labeller**
Type: `Labeller`.

**FVP_Base_Neoverse_E1.bp.virtio_rng**
VirtioEntropy device over MMIO transport.
Type: `VirtioEntropyMMIO`.

**FVP_Base_Neoverse_E1.bp.virtioblockdevice**
VirtioBlock device over MMIO transport.
Type: `VirtioBlockMMIO`.

**FVP_Base_Neoverse_E1.bp.virtioblockdevice_labeller**
Type: `Labeller`.

**FVP_Base_Neoverse_E1.bp.virtiop9device**
virtio P9 server.
Type: `VirtioP9Device`.

**FVP_Base_Neoverse_E1.bp.virtiop9device_labeller**
Type: `Labeller`.

**FVP_Base_Neoverse_E1.bp.vis**
Display window for VE using Visualisation library.
Type: `VEVisualisation`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Neoverse_E1.bp.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_Base_Neoverse_E1.bp.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_E1.bp.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Neoverse_E1.bp.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Neoverse_E1.bp.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Neoverse_E1.bp.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Neoverse_E1.bp.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_E1.bp.vram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Neoverse_E1.cci400**

Cache Coherent Interconnect for AXI4 ACE.
Type: `CCI400`.

**FVP_Base_Neoverse_E1.clockdivider0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_E1.cluster0**

ARM Neoverse-E1 Cluster CT model.
Type: `Cluster_ARM_Neoverse-E1`.

**FVP_Base_Neoverse_E1.cluster0.DSU.shared_cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Neoverse_E1.cluster0.cpu0.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Neoverse_E1.cluster0.cpu0.thread0**

ARM Neoverse-E1 CT model.
Type: `ARM_Neoverse-E1`.

**FVP_Base_Neoverse_E1.cluster0.cpu0.thread1**

ARM Neoverse-E1 CT model.
Type: `ARM_Neoverse-E1`.

**FVP_Base_Neoverse_E1.cluster0.cpu1.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Neoverse_E1.cluster0.cpu1.thread0**

ARM Neoverse-E1 CT model.
Type: `ARM_Neoverse-E1`.

**FVP_Base_Neoverse_E1.cluster0.cpu1.thread1**

ARM Neoverse-E1 CT model.
Type: `ARM_Neoverse-E1`.

**FVP_Base_Neoverse_E1.cluster0.cpu2.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Neoverse_E1.cluster0.cpu2.thread0**

ARM Neoverse-E1 CT model.
Type: `ARM_Neoverse-E1`.

**FVP_Base_Neoverse_E1.cluster0.cpu2.thread1**

ARM Neoverse-E1 CT model.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ARM_Neoverse-E1`.

**FVP_Base_Neoverse_E1.cluster0.cpu3.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Neoverse_E1.cluster0.cpu3.thread0**

ARM Neoverse-E1 CT model.
Type: `ARM_Neoverse-E1`.

**FVP_Base_Neoverse_E1.cluster0.cpu3.thread1**

ARM Neoverse-E1 CT model.
Type: `ARM_Neoverse-E1`.

**FVP_Base_Neoverse_E1.cluster0.cpu4.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Neoverse_E1.cluster0.cpu4.thread0**

ARM Neoverse-E1 CT model.
Type: `ARM_Neoverse-E1`.

**FVP_Base_Neoverse_E1.cluster0.cpu4.thread1**

ARM Neoverse-E1 CT model.
Type: `ARM_Neoverse-E1`.

**FVP_Base_Neoverse_E1.cluster0.cpu5.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Neoverse_E1.cluster0.cpu5.thread0**

ARM Neoverse-E1 CT model.
Type: `ARM_Neoverse-E1`.

**FVP_Base_Neoverse_E1.cluster0.cpu5.thread1**

ARM Neoverse-E1 CT model.
Type: `ARM_Neoverse-E1`.

**FVP_Base_Neoverse_E1.cluster0.cpu6.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Neoverse_E1.cluster0.cpu6.thread0**

ARM Neoverse-E1 CT model.
Type: `ARM_Neoverse-E1`.

**FVP_Base_Neoverse_E1.cluster0.cpu6.thread1**

ARM Neoverse-E1 CT model.
Type: `ARM_Neoverse-E1`.

**FVP_Base_Neoverse_E1.cluster0.cpu7.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Neoverse_E1.cluster0.cpu7.thread0**

ARM Neoverse-E1 CT model.
Type: `ARM_Neoverse-E1`.

**FVP_Base_Neoverse_E1.cluster0.cpu7.thread1**

ARM Neoverse-E1 CT model.
Type: `ARM_Neoverse-E1`.

**FVP_Base_Neoverse_E1.cluster0_labeller**

Type: `Labeller`.

**FVP_Base_Neoverse_E1.dapmemlogger**

Bus Logger.
Type: `PVBusLogger`.

**FVP_Base_Neoverse_E1.elfloader**

ELF loader component.
Type: `ElfLoader`.

**FVP_Base_Neoverse_E1.gic_distributor**

GIC Interrupt Redistribution Infrastucture component.
Type: `GIC_IRI`.

**FVP_Base_Neoverse_E1.pctl**

Base Platforms Power Controller.
Type: `Base_PowerController`.

**FVP_Base_Neoverse_E1.pctl.timer_reset**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Neoverse_E1.pctl.timer_reset.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Neoverse_E1.pctl.timer_reset.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Neoverse_E1.pctl.timer_reset.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

# 13.69 FVP_Base_Neoverse-N1

List of instances in FVP_Base_Neoverse-N1.

## FVP_Base_Neoverse-N1 instances

**FVP_Base_Neoverse_N1**

Base Platform Compute Subsystem for ARMNeoverseN1CT.
Type: `FVP_Base_Neoverse_N1`.

**FVP_Base_Neoverse_N1.bp**

Peripherals and address map for the Base Platform.
Type: `BasePlatformPeripherals`.

**FVP_Base_Neoverse_N1.bp.Timer_0_1**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Neoverse_N1.bp.Timer_0_1.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_N1.bp.Timer_0_1.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_N1.bp.Timer_0_1.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Neoverse_N1.bp.Timer_0_1.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Neoverse_N1.bp.Timer_2_3**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Neoverse_N1.bp.Timer_2_3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_N1.bp.Timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_N1.bp.Timer_2_3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Neoverse_N1.bp.Timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Neoverse_N1.bp.ap_refclk**

ARM Generic Timer.
Type: `MemoryMappedGenericTimer`.

**FVP_Base_Neoverse_N1.bp.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**FVP_Base_Neoverse_N1.bp.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_N1.bp.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_N1.bp.clock300MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Neoverse_N1.bp.clock32KHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_N1.bp.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_N1.bp.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_N1.bp.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_N1.bp.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_N1.bp.dmc**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Neoverse_N1.bp.dmc_phy**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Neoverse_N1.bp.dummy_local_dap_rom**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Neoverse_N1.bp.dummy_ram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Neoverse_N1.bp.dummy_usb**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Neoverse_N1.bp.exclusive_monitor**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_Base_Neoverse_N1.bp.flash0**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Neoverse_N1.bp.flash1**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Neoverse_N1.bp.flashloader0**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Neoverse_N1.bp.flashloader1**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Neoverse_N1.bp.generic_watchdog**

ARM Generic Watchdog.
Type: `MemoryMappedGenericWatchdog`.

**FVP_Base_Neoverse_N1.bp.hdlcd0**

ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).
Type: `PL370_HDLCD`.

**FVP_Base_Neoverse_N1.bp.hdlcd0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Neoverse_N1.bp.hdlcd0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Neoverse_N1.bp.hdlcd0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Neoverse_N1.bp.hdlcd0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Neoverse_N1.bp.hdlcd0_labeller**

Type: `Labeller`.

**FVP_Base_Neoverse_N1.bp.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_Base_Neoverse_N1.bp.mmc**

Generic Multimedia Card.
Type: `MMC`.

**FVP_Base_Neoverse_N1.bp.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Neoverse_N1.bp.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Neoverse_N1.bp.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Neoverse_N1.bp.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Neoverse_N1.bp.mpe**

Encrypt memory transactions for each encryption context with an independent key to prevent mismatch access.
Type: `PVMemoryProtectionEngine`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Neoverse_N1.bp.nontrustedrom**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Neoverse_N1.bp.nontrustedromloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Neoverse_N1.bp.ns_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Neoverse_N1.bp.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Neoverse_N1.bp.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_N1.bp.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Neoverse_N1.bp.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Neoverse_N1.bp.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Neoverse_N1.bp.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Neoverse_N1.bp.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Neoverse_N1.bp.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_N1.bp.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Neoverse_N1.bp.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Neoverse_N1.bp.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Neoverse_N1.bp.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Neoverse_N1.bp.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Neoverse_N1.bp.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Neoverse_N1.bp.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Neoverse_N1.bp.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Neoverse_N1.bp.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Neoverse_N1.bp.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Neoverse_N1.bp.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Neoverse_N1.bp.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_N1.bp.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Neoverse_N1.bp.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Neoverse_N1.bp.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Neoverse_N1.bp.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Neoverse_N1.bp.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

**FVP_Base_Neoverse_N1.bp.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Neoverse_N1.bp.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Neoverse_N1.bp.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Neoverse_N1.bp.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Neoverse_N1.bp.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_Base_Neoverse_N1.bp.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Neoverse_N1.bp.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Neoverse_N1.bp.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Neoverse_N1.bp.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Neoverse_N1.bp.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Neoverse_N1.bp.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_N1.bp.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Neoverse_N1.bp.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_N1.bp.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_Base_Neoverse_N1.bp.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

**FVP_Base_Neoverse_N1.bp.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Neoverse_N1.bp.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Neoverse_N1.bp.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Neoverse_N1.bp.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Neoverse_N1.bp.pl111_clcd_labeller**

Type: `Labeller`.

**FVP_Base_Neoverse_N1.bp.pl180_mci**

ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

**FVP_Base_Neoverse_N1.bp.ps2keyboard**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Neoverse_N1.bp.ps2keyboard.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Neoverse_N1.bp.ps2keyboard.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Neoverse_N1.bp.ps2keyboard.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Neoverse_N1.bp.ps2keyboard.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Neoverse_N1.bp.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

**FVP_Base_Neoverse_N1.bp.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Neoverse_N1.bp.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Neoverse_N1.bp.ps2mouse.ps2_clocktimer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Neoverse_N1.bp.ps2mouse.ps2_clocktimer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Neoverse_N1.bp.psram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Neoverse_N1.bp.refcounter**
Memory Mapped Counter Module for Generic Timers.
Type: `MemoryMappedCounterModule`.

**FVP_Base_Neoverse_N1.bp.reset_or**
Or Gate.
Type: `OrGate`.

**FVP_Base_Neoverse_N1.bp.rl_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Neoverse_N1.bp.rt_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Neoverse_N1.bp.s_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Neoverse_N1.bp.secureDRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Neoverse_N1.bp.secureSRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Neoverse_N1.bp.secureflash**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Neoverse_N1.bp.secureflashloader**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Neoverse_N1.bp.smsc_91c111**

SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_Base_Neoverse_N1.bp.sp805_wdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Neoverse_N1.bp.sp805_wdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Neoverse_N1.bp.sp805_wdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Neoverse_N1.bp.sp805_wdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Neoverse_N1.bp.sp805_wdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Neoverse_N1.bp.sp810_sysctrl**

Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_Base_Neoverse_N1.bp.sp810_sysctrl.clkdiv_clk0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_N1.bp.sp810_sysctrl.clkdiv_clk1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_N1.bp.sp810_sysctrl.clkdiv_clk2**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_N1.bp.sp810_sysctrl.clkdiv_clk3**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_N1.bp.sram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Neoverse_N1.bp.terminal_0**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Neoverse_N1.bp.terminal_1**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Neoverse_N1.bp.terminal_2**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Neoverse_N1.bp.terminal_3**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Neoverse_N1.bp.trusted_key_storage**

Trusted Root-Key Storage unit.
Type: `RootKeyStorage`.

**FVP_Base_Neoverse_N1.bp.trusted_nv_counter**

Trusted Non-Volatile Counter unit.
Type: `NonVolatileCounter`.

**FVP_Base_Neoverse_N1.bp.trusted_rng**

Random Number Generator unit.
Type: `RandomNumberGenerator`.

**FVP_Base_Neoverse_N1.bp.trusted_watchdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Neoverse_N1.bp.trusted_watchdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Neoverse_N1.bp.trusted_watchdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Neoverse_N1.bp.trusted_watchdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Neoverse_N1.bp.trusted_watchdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Neoverse_N1.bp.tzc_400**

TrustZone Address Space Controller.
Type: `TZC_400`.

**FVP_Base_Neoverse_N1.bp.ve_sysregs**

Type: `VE_SysRegs`.

**FVP_Base_Neoverse_N1.bp.vedcc**

Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

**FVP_Base_Neoverse_N1.bp.virtio_net**

VirtioNet device over MMIO transport.
Type: `VirtioNetMMIO`.

**FVP_Base_Neoverse_N1.bp.virtio_net_labeller**

Type: `Labeller`.

**FVP_Base_Neoverse_N1.bp.virtio_rng**

VirtioEntropy device over MMIO transport.
Type: `VirtioEntropyMMIO`.

**FVP_Base_Neoverse_N1.bp.virtioblockdevice**

VirtioBlock device over MMIO transport.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `VirtioBlockMMIO`.

**FVP_Base_Neoverse_N1.bp.virtioblockdevice_labeller**

Type: `Labeller`.

**FVP_Base_Neoverse_N1.bp.virtiop9device**

virtio P9 server.
Type: `VirtioP9Device`.

**FVP_Base_Neoverse_N1.bp.virtiop9device_labeller**

Type: `Labeller`.

**FVP_Base_Neoverse_N1.bp.vis**

Display window for VE using Visualisation library.
Type: `VEVisualisation`.

**FVP_Base_Neoverse_N1.bp.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_Base_Neoverse_N1.bp.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_N1.bp.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Neoverse_N1.bp.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Neoverse_N1.bp.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Neoverse_N1.bp.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Neoverse_N1.bp.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_N1.bp.vram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Neoverse_N1.cci400**

Cache Coherent Interconnect for AXI4 ACE.
Type: `CCI400`.

**FVP_Base_Neoverse_N1.clockdivider0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_N1.cluster0**

ARM Neoverse-N1 Cluster CT model.
Type: `Cluster_ARM_Neoverse-N1`.

**FVP_Base_Neoverse_N1.cluster0.DSU.shared_cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Neoverse_N1.cluster0.cpu0**

ARM Neoverse-N1 CT model.
Type: `ARM_Neoverse-N1`.

**FVP_Base_Neoverse_N1.cluster0.cpu0.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Neoverse_N1.cluster0.cpu0.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Neoverse_N1.cluster0.cpu0.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Neoverse_N1.cluster0.cpu0.l2cache**

PV Cache.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `PVCache`.

**FVP_Base_Neoverse_N1.cluster0.cpu1**

ARM Neoverse-N1 CT model.
Type: `ARM_Neoverse-N1`.

**FVP_Base_Neoverse_N1.cluster0.cpu1.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Neoverse_N1.cluster0.cpu1.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Neoverse_N1.cluster0.cpu1.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Neoverse_N1.cluster0.cpu1.l2cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Neoverse_N1.cluster0.cpu2**

ARM Neoverse-N1 CT model.
Type: `ARM_Neoverse-N1`.

**FVP_Base_Neoverse_N1.cluster0.cpu2.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Neoverse_N1.cluster0.cpu2.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Neoverse_N1.cluster0.cpu2.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Neoverse_N1.cluster0.cpu2.l2cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Neoverse_N1.cluster0.cpu3**

ARM Neoverse-N1 CT model.
Type: `ARM_Neoverse-N1`.

**FVP_Base_Neoverse_N1.cluster0.cpu3.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Neoverse_N1.cluster0.cpu3.l1dcache**

PV Cache.
Type: `PVCache`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Neoverse_N1.cluster0.cpu3.l1icache**

PV Cache.
Type: PVCache.

**FVP_Base_Neoverse_N1.cluster0.cpu3.l2cache**

PV Cache.
Type: PVCache.

**FVP_Base_Neoverse_N1.cluster0_labeller**

Type: Labeller.

**FVP_Base_Neoverse_N1.dapmemlogger**

Bus Logger.
Type: PVBusLogger.

**FVP_Base_Neoverse_N1.elfloader**

ELF loader component.
Type: ElfLoader.

**FVP_Base_Neoverse_N1.gic_distributor**

GIC Interrupt Redistribution Infrastucture component.
Type: GIC_IRI.

**FVP_Base_Neoverse_N1.pctl**

Base Platforms Power Controller.
Type: Base_PowerController.

**FVP_Base_Neoverse_N1.pctl.timer_reset**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: ClockTimerThread.

**FVP_Base_Neoverse_N1.pctl.timer_reset.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: ClockTimerThread64.

**FVP_Base_Neoverse_N1.pctl.timer_reset.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: SchedulerThread.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Neoverse_N1.pctl.timer_reset.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

# 13.70 FVP_Base_Neoverse-N2

List of instances in FVP_Base_Neoverse-N2.

## FVP_Base_Neoverse-N2 instances

**FVP_Base_Neoverse_N2**

Base Platform Compute Subsystem for ARMNeoverseN2CT.
Type: `FVP_Base_Neoverse_N2`.

**FVP_Base_Neoverse_N2.bp**

Peripherals and address map for the Base Platform.
Type: `BasePlatformPeripherals`.

**FVP_Base_Neoverse_N2.bp.Timer_0_1**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Neoverse_N2.bp.Timer_0_1.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_N2.bp.Timer_0_1.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_N2.bp.Timer_0_1.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Neoverse_N2.bp.Timer_0_1.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Neoverse_N2.bp.Timer_2_3**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Neoverse_N2.bp.Timer_2_3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_N2.bp.Timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_N2.bp.Timer_2_3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Neoverse_N2.bp.Timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Neoverse_N2.bp.ap_refclk**

ARM Generic Timer.
Type: `MemoryMappedGenericTimer`.

**FVP_Base_Neoverse_N2.bp.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**FVP_Base_Neoverse_N2.bp.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_N2.bp.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_N2.bp.clock300MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Neoverse_N2.bp.clock32KHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_N2.bp.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_N2.bp.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_N2.bp.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_N2.bp.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_N2.bp.dmc**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Neoverse_N2.bp.dmc_phy**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Neoverse_N2.bp.dummy_local_dap_rom**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Neoverse_N2.bp.dummy_ram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Neoverse_N2.bp.dummy_usb**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Neoverse_N2.bp.exclusive_monitor**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_Base_Neoverse_N2.bp.flash0**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Neoverse_N2.bp.flash1**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Neoverse_N2.bp.flashloader0**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Neoverse_N2.bp.flashloader1**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Neoverse_N2.bp.generic_watchdog**

ARM Generic Watchdog.
Type: `MemoryMappedGenericWatchdog`.

**FVP_Base_Neoverse_N2.bp.hdlcd0**

ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).
Type: `PL370_HDLCD`.

**FVP_Base_Neoverse_N2.bp.hdlcd0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Neoverse_N2.bp.hdlcd0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Neoverse_N2.bp.hdlcd0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Neoverse_N2.bp.hdlcd0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Neoverse_N2.bp.hdlcd0_labeller**

Type: `Labeller`.

**FVP_Base_Neoverse_N2.bp.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_Base_Neoverse_N2.bp.mmc**

Generic Multimedia Card.
Type: `MMC`.

**FVP_Base_Neoverse_N2.bp.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Neoverse_N2.bp.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Neoverse_N2.bp.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Neoverse_N2.bp.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Neoverse_N2.bp.mpe**

Encrypt memory transactions for each encryption context with an independent key to prevent mismatch access.
Type: `PVMemoryProtectionEngine`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Neoverse_N2.bp.nontrustedrom**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Neoverse_N2.bp.nontrustedromloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Neoverse_N2.bp.ns_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Neoverse_N2.bp.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Neoverse_N2.bp.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_N2.bp.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Neoverse_N2.bp.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Neoverse_N2.bp.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Neoverse_N2.bp.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Neoverse_N2.bp.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Neoverse_N2.bp.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_N2.bp.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Neoverse_N2.bp.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Neoverse_N2.bp.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Neoverse_N2.bp.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Neoverse_N2.bp.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Neoverse_N2.bp.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Neoverse_N2.bp.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Neoverse_N2.bp.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Neoverse_N2.bp.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Neoverse_N2.bp.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Neoverse_N2.bp.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Neoverse_N2.bp.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_N2.bp.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Neoverse_N2.bp.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Neoverse_N2.bp.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Neoverse_N2.bp.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Neoverse_N2.bp.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

**FVP_Base_Neoverse_N2.bp.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Neoverse_N2.bp.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Neoverse_N2.bp.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Neoverse_N2.bp.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Neoverse_N2.bp.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_Base_Neoverse_N2.bp.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Neoverse_N2.bp.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Neoverse_N2.bp.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Neoverse_N2.bp.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Neoverse_N2.bp.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Neoverse_N2.bp.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_N2.bp.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Neoverse_N2.bp.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_N2.bp.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_Base_Neoverse_N2.bp.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

**FVP_Base_Neoverse_N2.bp.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Neoverse_N2.bp.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Neoverse_N2.bp.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Neoverse_N2.bp.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Neoverse_N2.bp.pl111_clcd_labeller**

Type: `Labeller`.

**FVP_Base_Neoverse_N2.bp.pl180_mci**

ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

**FVP_Base_Neoverse_N2.bp.ps2keyboard**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Neoverse_N2.bp.ps2keyboard.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Neoverse_N2.bp.ps2keyboard.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Neoverse_N2.bp.ps2keyboard.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Neoverse_N2.bp.ps2keyboard.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Neoverse_N2.bp.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

**FVP_Base_Neoverse_N2.bp.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Neoverse_N2.bp.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Neoverse_N2.bp.ps2mouse.ps2_clocktimer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Neoverse_N2.bp.ps2mouse.ps2_clocktimer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Neoverse_N2.bp.psram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Neoverse_N2.bp.refcounter**
Memory Mapped Counter Module for Generic Timers.
Type: `MemoryMappedCounterModule`.

**FVP_Base_Neoverse_N2.bp.reset_or**
Or Gate.
Type: `OrGate`.

**FVP_Base_Neoverse_N2.bp.rl_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Neoverse_N2.bp.rt_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Neoverse_N2.bp.s_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Neoverse_N2.bp.secureDRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Neoverse_N2.bp.secureSRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Neoverse_N2.bp.secureflash**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Neoverse_N2.bp.secureflashloader**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Neoverse_N2.bp.smsc_91c111**

SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_Base_Neoverse_N2.bp.sp805_wdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Neoverse_N2.bp.sp805_wdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Neoverse_N2.bp.sp805_wdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Neoverse_N2.bp.sp805_wdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Neoverse_N2.bp.sp805_wdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Neoverse_N2.bp.sp810_sysctrl**

Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_Base_Neoverse_N2.bp.sp810_sysctrl.clkdiv_clk0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_N2.bp.sp810_sysctrl.clkdiv_clk1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_N2.bp.sp810_sysctrl.clkdiv_clk2**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_N2.bp.sp810_sysctrl.clkdiv_clk3**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_N2.bp.sram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Neoverse_N2.bp.terminal_0**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Neoverse_N2.bp.terminal_1**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Neoverse_N2.bp.terminal_2**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Neoverse_N2.bp.terminal_3**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Neoverse_N2.bp.trusted_key_storage**

Trusted Root-Key Storage unit.
Type: `RootKeyStorage`.

**FVP_Base_Neoverse_N2.bp.trusted_nv_counter**

Trusted Non-Volatile Counter unit.
Type: `NonVolatileCounter`.

**FVP_Base_Neoverse_N2.bp.trusted_rng**

Random Number Generator unit.
Type: `RandomNumberGenerator`.

**FVP_Base_Neoverse_N2.bp.trusted_watchdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Neoverse_N2.bp.trusted_watchdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Neoverse_N2.bp.trusted_watchdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Neoverse_N2.bp.trusted_watchdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Neoverse_N2.bp.trusted_watchdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Neoverse_N2.bp.tzc_400**

TrustZone Address Space Controller.
Type: `TZC_400`.

**FVP_Base_Neoverse_N2.bp.ve_sysregs**

Type: `VE_SysRegs`.

**FVP_Base_Neoverse_N2.bp.vedcc**

Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

**FVP_Base_Neoverse_N2.bp.virtio_net**

VirtioNet device over MMIO transport.
Type: `VirtioNetMMIO`.

**FVP_Base_Neoverse_N2.bp.virtio_net_labeller**

Type: `Labeller`.

**FVP_Base_Neoverse_N2.bp.virtio_rng**

VirtioEntropy device over MMIO transport.
Type: `VirtioEntropyMMIO`.

**FVP_Base_Neoverse_N2.bp.virtioblockdevice**

VirtioBlock device over MMIO transport.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `VirtioBlockMMIO`.

**FVP_Base_Neoverse_N2.bp.virtioblockdevice_labeller**

Type: `Labeller`.

**FVP_Base_Neoverse_N2.bp.virtiop9device**

virtio P9 server.
Type: `VirtioP9Device`.

**FVP_Base_Neoverse_N2.bp.virtiop9device_labeller**

Type: `Labeller`.

**FVP_Base_Neoverse_N2.bp.vis**

Display window for VE using Visualisation library.
Type: `VEVisualisation`.

**FVP_Base_Neoverse_N2.bp.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_Base_Neoverse_N2.bp.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_N2.bp.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Neoverse_N2.bp.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Neoverse_N2.bp.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Neoverse_N2.bp.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Neoverse_N2.bp.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_N2.bp.vram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Neoverse_N2.cci400**

Cache Coherent Interconnect for AXI4 ACE.
Type: `CCI400`.

**FVP_Base_Neoverse_N2.clockdivider0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_N2.cluster0**

ARM Neoverse-N2 Cluster CT model.
Type: `Cluster_ARM_Neoverse-N2`.

**FVP_Base_Neoverse_N2.cluster0.DSU.shared_cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Neoverse_N2.cluster0.cpu0**

ARM Neoverse-N2 CT model.
Type: `ARM_Neoverse-N2`.

**FVP_Base_Neoverse_N2.cluster0.cpu0.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Neoverse_N2.cluster0.cpu0.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Neoverse_N2.cluster0.cpu0.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Neoverse_N2.cluster0.cpu0.l2cache**

PV Cache.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `PVCache`.

**FVP_Base_Neoverse_N2.cluster0_labeller**

Type: `Labeller`.

**FVP_Base_Neoverse_N2.dapmemlogger**

Bus Logger.
Type: `PVBusLogger`.

**FVP_Base_Neoverse_N2.elfloader**

ELF loader component.
Type: `ElfLoader`.

**FVP_Base_Neoverse_N2.gic_distributor**

GIC Interrupt Redistribution Infrastucture component.
Type: `GIC_IRI`.

**FVP_Base_Neoverse_N2.pctl**

Base Platforms Power Controller.
Type: `Base_PowerController`.

**FVP_Base_Neoverse_N2.pctl.timer_reset**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Neoverse_N2.pctl.timer_reset.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Neoverse_N2.pctl.timer_reset.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Neoverse_N2.pctl.timer_reset.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

# 13.71 FVP_Base_Neoverse-N2x1-Neoverse-N2x1

List of instances in FVP_Base_Neoverse-N2x1-Neoverse-N2x1.

## FVP_Base_Neoverse-N2x1-Neoverse-N2x1 instances

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1**

Base Platform Compute Subsystem for ARMNeoverseN2x1CT and ARMNeoverseN2x1CT.
Type: `FVP_Base_Neoverse_N2x1_Neoverse_N2x1`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp**

Peripherals and address map for the Base Platform.
Type: `BasePlatformPeripherals`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.Timer_0_1**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.Timer_0_1.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.Timer_0_1.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.Timer_0_1.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.Timer_0_1.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.Timer_2_3**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.Timer_2_3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.Timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.Timer_2_3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.Timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.ap_refclk**

ARM Generic Timer.
Type: `MemoryMappedGenericTimer`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.clock300MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.clock32KHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.dmc**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.dmc_phy**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.dummy_local_dap_rom**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.dummy_ram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.dummy_usb**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.exclusive_monitor**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.flash0**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.flash1**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.flashloader0**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.flashloader1**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.generic_watchdog**

ARM Generic Watchdog.
Type: `MemoryMappedGenericWatchdog`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.hdlcd0**

ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).
Type: `PL370_HDLCD`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.hdlcd0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.hdlcd0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.hdlcd0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.hdlcd0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.hdlcd0_labeller**

Type: `Labeller`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.mmc**

Generic Multimedia Card.
Type: `MMC`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.mpe**

Encrypt memory transactions for each encryption context with an independent key to prevent mismatch access.
Type: `PVMemoryProtectionEngine`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.nontrustedrom**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.nontrustedromloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.ns_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ClockTimerThread`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.pl111_clcd_labeller**

Type: `Labeller`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.pl180_mci**

ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.ps2keyboard**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.ps2keyboard.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.ps2keyboard.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.ps2keyboard.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.ps2keyboard.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.ps2mouse.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.ps2mouse.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.psram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.refcounter**
Memory Mapped Counter Module for Generic Timers.
Type: `MemoryMappedCounterModule`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.reset_or**
Or Gate.
Type: `OrGate`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.rl_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.rt_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.s_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.secureDRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.secureSRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.secureflash**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.secureflashloader**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.smsc_91c111**
SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.sp805_wdog**
ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.sp805_wdog.clocktimer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.sp805_wdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.sp805_wdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.sp805_wdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.sp810_sysctrl**

Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.sp810_sysctrl.clkdiv_clk0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.sp810_sysctrl.clkdiv_clk1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.sp810_sysctrl.clkdiv_clk2**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.sp810_sysctrl.clkdiv_clk3**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.sram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.terminal_0**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.terminal_1**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.terminal_2**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.terminal_3**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.trusted_key_storage**
Trusted Root-Key Storage unit.
Type: `RootKeyStorage`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.trusted_nv_counter**
Trusted Non-Volatile Counter unit.
Type: `NonVolatileCounter`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.trusted_rng**
Random Number Generator unit.
Type: `RandomNumberGenerator`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.trusted_watchdog**
ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.trusted_watchdog.clocktimer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.trusted_watchdog.clocktimer.timer**
A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.trusted_watchdog.clocktimer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.trusted_watchdog.clocktimer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.tzc_400**
TrustZone Address Space Controller.
Type: `TZC_400`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.ve_sysregs**
Type: `VE_SysRegs`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.vedcc**
Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.virtio_net**
VirtioNet device over MMIO transport.
Type: `VirtioNetMMIO`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.virtio_net_labeller**
Type: `Labeller`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.virtio_rng**
VirtioEntropy device over MMIO transport.
Type: `VirtioEntropyMMIO`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.virtioblockdevice**
VirtioBlock device over MMIO transport.
Type: `VirtioBlockMMIO`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.virtioblockdevice_labeller**
Type: `Labeller`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.virtiop9device**
virtio P9 server.
Type: `VirtioP9Device`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.virtiop9device_labeller**
Type: `Labeller`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.vis**
Display window for VE using Visualisation library.
Type: `VEVisualisation`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.bp.vram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.cci400**

Cache Coherent Interconnect for AXI4 ACE.
Type: `CCI400`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.clockdivider0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.clockdivider1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.cluster0**

ARM Neoverse-N2 Cluster CT model.
Type: `Cluster_ARM_Neoverse-N2`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.cluster0.DSU.shared_cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.cluster0.cpu0**

ARM Neoverse-N2 CT model.
Type: `ARM_Neoverse-N2`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.cluster0.cpu0.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.cluster0.cpu0.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.cluster0.cpu0.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.cluster0.cpu0.l2cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.cluster0_labeller**

Type: `Labeller`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.cluster1**

ARM Neoverse-N2 Cluster CT model.
Type: `Cluster_ARM_Neoverse-N2`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.cluster1.DSU.shared_cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.cluster1.cpu0**

ARM Neoverse-N2 CT model.
Type: `ARM_Neoverse-N2`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.cluster1.cpu0.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.cluster1.cpu0.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.cluster1.cpu0.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.cluster1.cpu0.l2cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.cluster1_labeller**

Type: `Labeller`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.dapmemlogger**

Bus Logger.
Type: `PVBusLogger`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.elfloader**

ELF loader component.
Type: `ElfLoader`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.gic_distributor**

GIC Interrupt Redistribution Infrastucture component.
Type: `GIC_IRI`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.pctl**

Base Platforms Power Controller.
Type: `Base_PowerController`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.pctl.timer_reset**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.pctl.timer_reset.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.pctl.timer_reset.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Neoverse_N2x1_Neoverse_N2x1.pctl.timer_reset.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

## 13.72 FVP_Base_Neoverse-V1

List of instances in FVP_Base_Neoverse-V1.

### FVP_Base_Neoverse-V1 instances

**FVP_Base_Neoverse_V1**

Base Platform Compute Subsystem for ARMNeoverseV1CT.
Type: `FVP_Base_Neoverse_V1`.

**FVP_Base_Neoverse_V1.bp**

Peripherals and address map for the Base Platform.
Type: `BasePlatformPeripherals`.

**FVP_Base_Neoverse_V1.bp.Timer_0_1**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Neoverse_V1.bp.Timer_0_1.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_V1.bp.Timer_0_1.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Neoverse_V1.bp.Timer_0_1.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Neoverse_V1.bp.Timer_0_1.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Neoverse_V1.bp.Timer_2_3**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_Neoverse_V1.bp.Timer_2_3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_V1.bp.Timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_V1.bp.Timer_2_3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Neoverse_V1.bp.Timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Neoverse_V1.bp.ap_refclk**

ARM Generic Timer.
Type: `MemoryMappedGenericTimer`.

**FVP_Base_Neoverse_V1.bp.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**FVP_Base_Neoverse_V1.bp.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_V1.bp.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**`FVP_Base_Neoverse_V1.bp.clock300MHz`**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**`FVP_Base_Neoverse_V1.bp.clock32KHz`**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**`FVP_Base_Neoverse_V1.bp.clock35MHz`**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**`FVP_Base_Neoverse_V1.bp.clock50Hz`**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**`FVP_Base_Neoverse_V1.bp.clockCLCD`**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**`FVP_Base_Neoverse_V1.bp.clockdivider`**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**`FVP_Base_Neoverse_V1.bp.dmc`**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Neoverse_V1.bp.dmc_phy**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Neoverse_V1.bp.dummy_local_dap_rom**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Neoverse_V1.bp.dummy_ram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Neoverse_V1.bp.dummy_usb**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Neoverse_V1.bp.exclusive_monitor**
Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_Base_Neoverse_V1.bp.flash0**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Neoverse_V1.bp.flash1**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Neoverse_V1.bp.flashloader0**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Neoverse_V1.bp.flashloader1**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Neoverse_V1.bp.generic_watchdog**
ARM Generic Watchdog.
Type: `MemoryMappedGenericWatchdog`.

**FVP_Base_Neoverse_V1.bp.hdlcd0**
ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).
Type: `PL370_HDLCD`.

**FVP_Base_Neoverse_V1.bp.hdlcd0.timer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Neoverse_V1.bp.hdlcd0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Neoverse_V1.bp.hdlcd0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Neoverse_V1.bp.hdlcd0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Neoverse_V1.bp.hdlcd0_labeller**

Type: `Labeller`.

**FVP_Base_Neoverse_V1.bp.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_Base_Neoverse_V1.bp.mmc**

Generic Multimedia Card.
Type: `MMC`.

**FVP_Base_Neoverse_V1.bp.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Neoverse_V1.bp.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Neoverse_V1.bp.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Neoverse_V1.bp.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Neoverse_V1.bp.mpe**

Encrypt memory transactions for each encryption context with an independent key to prevent mismatch access.
Type: `PVMemoryProtectionEngine`.

**FVP_Base_Neoverse_V1.bp.nontrustedrom**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Neoverse_V1.bp.nontrustedromloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Neoverse_V1.bp.ns_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Neoverse_V1.bp.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Neoverse_V1.bp.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_V1.bp.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Neoverse_V1.bp.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Neoverse_V1.bp.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Neoverse_V1.bp.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Neoverse_V1.bp.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Neoverse_V1.bp.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_V1.bp.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Neoverse_V1.bp.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Neoverse_V1.bp.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Neoverse_V1.bp.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Neoverse_V1.bp.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Neoverse_V1.bp.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_V1.bp.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Neoverse_V1.bp.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Neoverse_V1.bp.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Neoverse_V1.bp.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Neoverse_V1.bp.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Neoverse_V1.bp.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_V1.bp.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Neoverse_V1.bp.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Neoverse_V1.bp.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Neoverse_V1.bp.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Neoverse_V1.bp.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

**FVP_Base_Neoverse_V1.bp.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Neoverse_V1.bp.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Neoverse_V1.bp.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Neoverse_V1.bp.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Neoverse_V1.bp.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_Base_Neoverse_V1.bp.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Neoverse_V1.bp.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Neoverse_V1.bp.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Neoverse_V1.bp.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Neoverse_V1.bp.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Neoverse_V1.bp.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_V1.bp.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Neoverse_V1.bp.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_V1.bp.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_Base_Neoverse_V1.bp.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

**FVP_Base_Neoverse_V1.bp.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Neoverse_V1.bp.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Neoverse_V1.bp.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Neoverse_V1.bp.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Neoverse_V1.bp.pl111_clcd_labeller**

Type: `Labeller`.

**FVP_Base_Neoverse_V1.bp.pl180_mci**

ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Neoverse_V1.bp.ps2keyboard**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

**FVP_Base_Neoverse_V1.bp.ps2keyboard.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Neoverse_V1.bp.ps2keyboard.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Neoverse_V1.bp.ps2keyboard.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Neoverse_V1.bp.ps2keyboard.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Neoverse_V1.bp.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

**FVP_Base_Neoverse_V1.bp.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Neoverse_V1.bp.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Neoverse_V1.bp.ps2mouse.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Neoverse_V1.bp.ps2mouse.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Neoverse_V1.bp.psram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Neoverse_V1.bp.refcounter**

Memory Mapped Counter Module for Generic Timers.
Type: `MemoryMappedCounterModule`.

**FVP_Base_Neoverse_V1.bp.reset_or**

Or Gate.
Type: `OrGate`.

**FVP_Base_Neoverse_V1.bp.rl_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Neoverse_V1.bp.rt_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Neoverse_V1.bp.s_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Neoverse_V1.bp.secureDRAM**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Neoverse_V1.bp.secureSRAM**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Neoverse_V1.bp.secureflash**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Neoverse_V1.bp.secureflashloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Neoverse_V1.bp.smsc_91c111**

SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_Base_Neoverse_V1.bp.sp805_wdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Neoverse_V1.bp.sp805_wdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Neoverse_V1.bp.sp805_wdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Neoverse_V1.bp.sp805_wdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Neoverse_V1.bp.sp805_wdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Neoverse_V1.bp.sp810_sysctrl**

Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_Base_Neoverse_V1.bp.sp810_sysctrl.clkdiv_clk0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Neoverse_V1.bp.sp810_sysctrl.clkdiv_clk1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_V1.bp.sp810_sysctrl.clkdiv_clk2**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_V1.bp.sp810_sysctrl.clkdiv_clk3**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_V1.bp.sram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Neoverse_V1.bp.terminal_0**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Neoverse_V1.bp.terminal_1**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Neoverse_V1.bp.terminal_2**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Neoverse_V1.bp.terminal_3**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Neoverse_V1.bp.trusted_key_storage**

Trusted Root-Key Storage unit.
Type: `RootKeyStorage`.

**FVP_Base_Neoverse_V1.bp.trusted_nv_counter**

Trusted Non-Volatile Counter unit.
Type: `NonVolatileCounter`.

**FVP_Base_Neoverse_V1.bp.trusted_rng**

Random Number Generator unit.
Type: `RandomNumberGenerator`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Neoverse_V1.bp.trusted_watchdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Neoverse_V1.bp.trusted_watchdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Neoverse_V1.bp.trusted_watchdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Neoverse_V1.bp.trusted_watchdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Neoverse_V1.bp.trusted_watchdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Neoverse_V1.bp.tzc_400**

TrustZone Address Space Controller.
Type: `TZC_400`.

**FVP_Base_Neoverse_V1.bp.ve_sysregs**

Type: `VE_SysRegs`.

**FVP_Base_Neoverse_V1.bp.vedcc**

Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

**FVP_Base_Neoverse_V1.bp.virtio_net**

VirtioNet device over MMIO transport.
Type: `VirtioNetMMIO`.

**FVP_Base_Neoverse_V1.bp.virtio_net_labeller**

Type: `Labeller`.

**FVP_Base_Neoverse_V1.bp.virtio_rng**

VirtioEntropy device over MMIO transport.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `VirtioEntropyMMIO`.

**FVP_Base_Neoverse_V1.bp.virtioblockdevice**

VirtioBlock device over MMIO transport.
Type: `VirtioBlockMMIO`.

**FVP_Base_Neoverse_V1.bp.virtioblockdevice_labeller**

Type: `Labeller`.

**FVP_Base_Neoverse_V1.bp.virtiop9device**

virtio P9 server.
Type: `VirtioP9Device`.

**FVP_Base_Neoverse_V1.bp.virtiop9device_labeller**

Type: `Labeller`.

**FVP_Base_Neoverse_V1.bp.vis**

Display window for VE using Visualisation library.
Type: `VEVisualisation`.

**FVP_Base_Neoverse_V1.bp.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_Base_Neoverse_V1.bp.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_V1.bp.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Neoverse_V1.bp.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Neoverse_V1.bp.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Neoverse_V1.bp.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Neoverse_V1.bp.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_V1.bp.vram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Neoverse_V1.cci400**

Cache Coherent Interconnect for AXI4 ACE.
Type: `CCI400`.

**FVP_Base_Neoverse_V1.clockdivider0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_V1.cluster0**

ARM Neoverse-V1 Cluster CT model.
Type: `Cluster_ARM_Neoverse-V1`.

**FVP_Base_Neoverse_V1.cluster0.DSU.shared_cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Neoverse_V1.cluster0.cpu0**

ARM Neoverse-V1 CT model.
Type: `ARM_Neoverse-V1`.

**FVP_Base_Neoverse_V1.cluster0.cpu0.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Neoverse_V1.cluster0.cpu0.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Neoverse_V1.cluster0.cpu0.l1icache**

PV Cache.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `PVCache`.

**`FVP_Base_Neoverse_V1.cluster0.cpu0.l2cache`**

PV Cache.
Type: `PVCache`.

**`FVP_Base_Neoverse_V1.cluster0.cpu1`**

ARM Neoverse-V1 CT model.
Type: `ARM_Neoverse-V1`.

**`FVP_Base_Neoverse_V1.cluster0.cpu1.dtlb`**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**`FVP_Base_Neoverse_V1.cluster0.cpu1.l1dcache`**

PV Cache.
Type: `PVCache`.

**`FVP_Base_Neoverse_V1.cluster0.cpu1.l1icache`**

PV Cache.
Type: `PVCache`.

**`FVP_Base_Neoverse_V1.cluster0.cpu1.l2cache`**

PV Cache.
Type: `PVCache`.

**`FVP_Base_Neoverse_V1.cluster0.cpu2`**

ARM Neoverse-V1 CT model.
Type: `ARM_Neoverse-V1`.

**`FVP_Base_Neoverse_V1.cluster0.cpu2.dtlb`**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**`FVP_Base_Neoverse_V1.cluster0.cpu2.l1dcache`**

PV Cache.
Type: `PVCache`.

**`FVP_Base_Neoverse_V1.cluster0.cpu2.l1icache`**

PV Cache.
Type: `PVCache`.

**`FVP_Base_Neoverse_V1.cluster0.cpu2.l2cache`**

PV Cache.
Type: `PVCache`.

**`FVP_Base_Neoverse_V1.cluster0.cpu3`**

ARM Neoverse-V1 CT model.
Type: `ARM_Neoverse-V1`.

**`FVP_Base_Neoverse_V1.cluster0.cpu3.dtlb`**

TLB - instruction, data or unified.
Type: `TlbCadi`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Neoverse_V1.cluster0.cpu3.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Neoverse_V1.cluster0.cpu3.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Neoverse_V1.cluster0.cpu3.l2cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Neoverse_V1.cluster0_labeller**

Type: `Labeller`.

**FVP_Base_Neoverse_V1.dapmemlogger**

Bus Logger.
Type: `PVBusLogger`.

**FVP_Base_Neoverse_V1.elfloader**

ELF loader component.
Type: `ElfLoader`.

**FVP_Base_Neoverse_V1.gic_distributor**

GIC Interrupt Redistribution Infrastucture component.
Type: `GIC_IRI`.

**FVP_Base_Neoverse_V1.pctl**

Base Platforms Power Controller.
Type: `Base_PowerController`.

**FVP_Base_Neoverse_V1.pctl.timer_reset**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Neoverse_V1.pctl.timer_reset.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Neoverse_V1.pctl.timer_reset.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `SchedulerThread`.

**`FVP_Base_Neoverse_V1.pctl.timer_reset.timer.thread_event`**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

## 13.73 FVP_Base_Neoverse-V2x1-Neoverse-V2x1

List of instances in FVP_Base_Neoverse-V2x1-Neoverse-V2x1.

### FVP_Base_Neoverse-V2x1-Neoverse-V2x1 instances

**`FVP_Base_Neoverse_V2x1_Neoverse_V2x1`**

Base Platform Compute Subsystem for ARMNeoverseV2x1CT and ARMNeoverseV2x1CT.
Type: `FVP_Base_Neoverse_V2x1_Neoverse_V2x1`.

**`FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp`**

Peripherals and address map for the Base Platform.
Type: `BasePlatformPeripherals`.

**`FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.Timer_0_1`**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**`FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.Timer_0_1.clk_div0`**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**`FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.Timer_0_1.clk_div1`**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**`FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.Timer_0_1.counter0`**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**`FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.Timer_0_1.counter1`**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**`FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.Timer_2_3`**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.Timer_2_3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.Timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.Timer_2_3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.Timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.ap_refclk**

ARM Generic Timer.
Type: `MemoryMappedGenericTimer`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.clock300MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.clock32KHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.dmc**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.dmc_phy**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.dummy_local_dap_rom**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.dummy_ram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.dummy_usb**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.exclusive_monitor**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.flash0**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.flash1**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.flashloader0**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.flashloader1**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.generic_watchdog**

ARM Generic Watchdog.
Type: `MemoryMappedGenericWatchdog`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.hdlcd0**

ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).
Type: `PL370_HDLCD`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.hdlcd0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.hdlcd0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.hdlcd0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.hdlcd0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.hdlcd0_labeller**

Type: `Labeller`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.mmc**

Generic Multimedia Card.
Type: `MMC`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.mpe**

Encrypt memory transactions for each encryption context with an independent key to prevent mismatch access.
Type: `PVMemoryProtectionEngine`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.nontrustedrom**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.nontrustedromloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.ns_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.pl111_clcd_labeller**

Type: `Labeller`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.pl180_mci**

ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.ps2keyboard**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.ps2keyboard.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.ps2keyboard.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.ps2keyboard.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.ps2keyboard.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.ps2mouse.ps2_clocktimer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.ps2mouse.ps2_clocktimer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.psram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.refcounter**
Memory Mapped Counter Module for Generic Timers.
Type: `MemoryMappedCounterModule`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.reset_or**
Or Gate.
Type: `OrGate`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.rl_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.rt_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.s_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.secureDRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.secureSRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.secureflash**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.secureflashloader**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.smsc_91c111**

SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.sp805_wdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.sp805_wdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.sp805_wdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.sp805_wdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.sp805_wdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.sp810_sysctrl**

Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.sp810_sysctrl.clkdiv_clk0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.sp810_sysctrl.clkdiv_clk1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.sp810_sysctrl.clkdiv_clk2**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.sp810_sysctrl.clkdiv_clk3**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.sram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.terminal_0**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.terminal_1**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.terminal_2**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.terminal_3**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.trusted_key_storage**

Trusted Root-Key Storage unit.
Type: `RootKeyStorage`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.trusted_nv_counter**

Trusted Non-Volatile Counter unit.
Type: `NonVolatileCounter`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.trusted_rng**

Random Number Generator unit.
Type: `RandomNumberGenerator`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.trusted_watchdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.trusted_watchdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.trusted_watchdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.trusted_watchdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.trusted_watchdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.tzc_400**

TrustZone Address Space Controller.
Type: `TZC_400`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.ve_sysregs**

Type: `VE_SysRegs`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.vedcc**

Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.virtio_net**

VirtioNet device over MMIO transport.
Type: `VirtioNetMMIO`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.virtio_net_labeller**

Type: `Labeller`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.virtio_rng**

VirtioEntropy device over MMIO transport.
Type: `VirtioEntropyMMIO`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.virtioblockdevice**

VirtioBlock device over MMIO transport.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `VirtioBlockMMIO`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.virtioblockdevice_labeller**

Type: `Labeller`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.virtiop9device**

virtio P9 server.
Type: `VirtioP9Device`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.virtiop9device_labeller**

Type: `Labeller`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.vis**

Display window for VE using Visualisation library.
Type: `VEVisualisation`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.bp.vram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.cci400**

Cache Coherent Interconnect for AXI4 ACE.
Type: `CCI400`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.clockdivider0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.clockdivider1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.cluster0**

ARM Neoverse-V2 Cluster CT model.
Type: `Cluster_ARM_Neoverse-V2`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.cluster0.DSU.shared_cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.cluster0.cpu0**

ARM Neoverse-V2 CT model.
Type: `ARM_Neoverse-V2`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.cluster0.cpu0.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.cluster0.cpu0.l1dcache**

PV Cache.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `PVCache`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.cluster0.cpu0.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.cluster0.cpu0.l2cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.cluster0_labeller**

Type: `Labeller`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.cluster1**

ARM Neoverse-V2 Cluster CT model.
Type: `Cluster_ARM_Neoverse-V2`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.cluster1.DSU.shared_cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.cluster1.cpu0**

ARM Neoverse-V2 CT model.
Type: `ARM_Neoverse-V2`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.cluster1.cpu0.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.cluster1.cpu0.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.cluster1.cpu0.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.cluster1.cpu0.l2cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.cluster1_labeller**

Type: `Labeller`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.dapmemlogger**

Bus Logger.
Type: `PVBusLogger`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.elfloader**

ELF loader component.
Type: `ElfLoader`.

**FVP_Base_Neoverse_V2x1_Neoverse_V2x1.gic_distributor**

GIC Interrupt Redistribution Infrastucture component.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `GIC_IRI`.

**`FVP_Base_Neoverse_V2x1_Neoverse_V2x1.pctl`**

Base Platforms Power Controller.
Type: `Base_PowerController`.

**`FVP_Base_Neoverse_V2x1_Neoverse_V2x1.pctl.timer_reset`**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**`FVP_Base_Neoverse_V2x1_Neoverse_V2x1.pctl.timer_reset.timer`**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**`FVP_Base_Neoverse_V2x1_Neoverse_V2x1.pctl.timer_reset.timer.thread`**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**`FVP_Base_Neoverse_V2x1_Neoverse_V2x1.pctl.timer_reset.timer.thread_event`**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

## 13.74 FVP_Base_RevC-2xAEMvA

List of instances in FVP_Base_RevC-2xAEMvA.

### FVP_Base_RevC-2xAEMvA instances

**`FVP_Base_RevC_2xAEMvA`**

Base Platform Compute Subsystem for AEMvACT and AEMvACT.
Type: `FVP_Base_RevC_2xAEMvA`.

**`FVP_Base_RevC_2xAEMvA.bp`**

Peripherals and address map for the Base Platform.
Type: `BasePlatformPeripherals`.

**`FVP_Base_RevC_2xAEMvA.bp.Timer_0_1`**

ARM Dual-Timer Module(SP804).

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `SP804_Timer`.

**FVP_Base_RevC_2xAEMvA.bp.Timer_0_1.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_RevC_2xAEMvA.bp.Timer_0_1.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_RevC_2xAEMvA.bp.Timer_0_1.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_RevC_2xAEMvA.bp.Timer_0_1.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_RevC_2xAEMvA.bp.Timer_2_3**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_Base_RevC_2xAEMvA.bp.Timer_2_3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_RevC_2xAEMvA.bp.Timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_RevC_2xAEMvA.bp.Timer_2_3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_RevC_2xAEMvA.bp.Timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_Base_RevC_2xAEMvA.bp.ap_refclk**

ARM Generic Timer.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `MemoryMappedGenericTimer`.

**FVP_Base_RevC_2xAEMvA.bp.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**FVP_Base_RevC_2xAEMvA.bp.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_RevC_2xAEMvA.bp.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_RevC_2xAEMvA.bp.clock300MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_RevC_2xAEMvA.bp.clock32KHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_RevC_2xAEMvA.bp.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_RevC_2xAEMvA.bp.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_RevC_2xAEMvA.bp.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_RevC_2xAEMvA.bp.clockdivider**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_RevC_2xAEMvA.bp.dmc**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_RevC_2xAEMvA.bp.dmc_phy**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_RevC_2xAEMvA.bp.dummy_local_dap_rom**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_RevC_2xAEMvA.bp.dummy_ram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_RevC_2xAEMvA.bp.dummy_usb**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_RevC_2xAEMvA.bp.exclusive_monitor**
Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_Base_RevC_2xAEMvA.bp.flash0**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_RevC_2xAEMvA.bp.flash1**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_RevC_2xAEMvA.bp.flashloader0**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_RevC_2xAEMvA.bp.flashloader1**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_RevC_2xAEMvA.bp.generic_watchdog**
ARM Generic Watchdog.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `MemoryMappedGenericWatchdog`.

**FVP_Base_RevC_2xAEMvA.bp.hdlcd0**

ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).
Type: `PL370_HDLCD`.

**FVP_Base_RevC_2xAEMvA.bp.hdlcd0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_RevC_2xAEMvA.bp.hdlcd0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_RevC_2xAEMvA.bp.hdlcd0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_RevC_2xAEMvA.bp.hdlcd0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_RevC_2xAEMvA.bp.hdlcd0_labeller**

Type: `Labeller`.

**FVP_Base_RevC_2xAEMvA.bp.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_Base_RevC_2xAEMvA.bp.mmc**

Generic Multimedia Card.
Type: `MMC`.

**FVP_Base_RevC_2xAEMvA.bp.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_RevC_2xAEMvA.bp.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_RevC_2xAEMvA.bp.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_RevC_2xAEMvA.bp.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_RevC_2xAEMvA.bp.mpe**

Encrypt memory transactions for each encryption context with an independent key to prevent mismatch access.
Type: `PVMemoryProtectionEngine`.

**FVP_Base_RevC_2xAEMvA.bp.nontrustedrom**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_RevC_2xAEMvA.bp.nontrustedromloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_RevC_2xAEMvA.bp.ns_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_RevC_2xAEMvA.bp.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_RevC_2xAEMvA.bp.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_RevC_2xAEMvA.bp.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_RevC_2xAEMvA.bp.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_RevC_2xAEMvA.bp.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_RevC_2xAEMvA.bp.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_RevC_2xAEMvA.bp.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_RevC_2xAEMvA.bp.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_RevC_2xAEMvA.bp.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_RevC_2xAEMvA.bp.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_RevC_2xAEMvA.bp.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_RevC_2xAEMvA.bp.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_RevC_2xAEMvA.bp.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_RevC_2xAEMvA.bp.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_RevC_2xAEMvA.bp.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_RevC_2xAEMvA.bp.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_RevC_2xAEMvA.bp.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_RevC_2xAEMvA.bp.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_RevC_2xAEMvA.bp.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_Base_RevC_2xAEMvA.bp.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_RevC_2xAEMvA.bp.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_RevC_2xAEMvA.bp.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_RevC_2xAEMvA.bp.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_RevC_2xAEMvA.bp.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_RevC_2xAEMvA.bp.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

**FVP_Base_RevC_2xAEMvA.bp.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal()

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_RevC_2xAEMvA.bp.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_RevC_2xAEMvA.bp.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_RevC_2xAEMvA.bp.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_RevC_2xAEMvA.bp.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_Base_RevC_2xAEMvA.bp.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_RevC_2xAEMvA.bp.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_RevC_2xAEMvA.bp.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_RevC_2xAEMvA.bp.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_RevC_2xAEMvA.bp.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_RevC_2xAEMvA.bp.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_RevC_2xAEMvA.bp.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_Base_RevC_2xAEMvA.bp.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_RevC_2xAEMvA.bp.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_Base_RevC_2xAEMvA.bp.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

**FVP_Base_RevC_2xAEMvA.bp.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_RevC_2xAEMvA.bp.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_RevC_2xAEMvA.bp.pl111_clcd.pl11x_clcd.timer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_RevC_2xAEMvA.bp.pl111_clcd.pl11x_clcd.timer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_RevC_2xAEMvA.bp.pl111_clcd_labeller**
Type: `Labeller`.

**FVP_Base_RevC_2xAEMvA.bp.pl180_mci**
ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

**FVP_Base_RevC_2xAEMvA.bp.ps2keyboard**
Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

**FVP_Base_RevC_2xAEMvA.bp.ps2keyboard.ps2_clocktimer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_RevC_2xAEMvA.bp.ps2keyboard.ps2_clocktimer.timer**
A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_RevC_2xAEMvA.bp.ps2keyboard.ps2_clocktimer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_RevC_2xAEMvA.bp.ps2keyboard.ps2_clocktimer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `SchedulerThreadEvent`.

**FVP_Base_RevC_2xAEMvA.bp.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

**FVP_Base_RevC_2xAEMvA.bp.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_RevC_2xAEMvA.bp.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_RevC_2xAEMvA.bp.ps2mouse.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_RevC_2xAEMvA.bp.ps2mouse.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_RevC_2xAEMvA.bp.psram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_RevC_2xAEMvA.bp.refcounter**

Memory Mapped Counter Module for Generic Timers.
Type: `MemoryMappedCounterModule`.

**FVP_Base_RevC_2xAEMvA.bp.reset_or**

Or Gate.
Type: `OrGate`.

**FVP_Base_RevC_2xAEMvA.bp.rl_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_RevC_2xAEMvA.bp.rt_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_RevC_2xAEMvA.bp.s_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_RevC_2xAEMvA.bp.secureDRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_RevC_2xAEMvA.bp.secureSRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_RevC_2xAEMvA.bp.secureflash**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_Base_RevC_2xAEMvA.bp.secureflashloader**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_Base_RevC_2xAEMvA.bp.smsc_91c111**
SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_Base_RevC_2xAEMvA.bp.sp805_wdog**
ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_RevC_2xAEMvA.bp.sp805_wdog.clocktimer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_RevC_2xAEMvA.bp.sp805_wdog.clocktimer.timer**
A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_RevC_2xAEMvA.bp.sp805_wdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_RevC_2xAEMvA.bp.sp805_wdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_RevC_2xAEMvA.bp.sp810_sysctrl**

Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_Base_RevC_2xAEMvA.bp.sp810_sysctrl.clkdiv_clk0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_RevC_2xAEMvA.bp.sp810_sysctrl.clkdiv_clk1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_RevC_2xAEMvA.bp.sp810_sysctrl.clkdiv_clk2**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_RevC_2xAEMvA.bp.sp810_sysctrl.clkdiv_clk3**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_RevC_2xAEMvA.bp.sram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_RevC_2xAEMvA.bp.terminal_0**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_RevC_2xAEMvA.bp.terminal_1**

Telnet terminal interface.
Type: `TelnetTerminal`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_RevC_2xAEMvA.bp.terminal_2**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_RevC_2xAEMvA.bp.terminal_3**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_Base_RevC_2xAEMvA.bp.trusted_key_storage**

Trusted Root-Key Storage unit.
Type: `RootKeyStorage`.

**FVP_Base_RevC_2xAEMvA.bp.trusted_nv_counter**

Trusted Non-Volatile Counter unit.
Type: `NonVolatileCounter`.

**FVP_Base_RevC_2xAEMvA.bp.trusted_rng**

Random Number Generator unit.
Type: `RandomNumberGenerator`.

**FVP_Base_RevC_2xAEMvA.bp.trusted_watchdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_Base_RevC_2xAEMvA.bp.trusted_watchdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_RevC_2xAEMvA.bp.trusted_watchdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_RevC_2xAEMvA.bp.trusted_watchdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_RevC_2xAEMvA.bp.trusted_watchdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_RevC_2xAEMvA.bp.tzc_400**

TrustZone Address Space Controller.
Type: `TZC_400`.

**FVP_Base_RevC_2xAEMvA.bp.ve_sysregs**

Type: `VE_SysRegs`.

**FVP_Base_RevC_2xAEMvA.bp.vedcc**

Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

**FVP_Base_RevC_2xAEMvA.bp.virtio_net**

VirtioNet device over MMIO transport.
Type: `VirtioNetMMIO`.

**FVP_Base_RevC_2xAEMvA.bp.virtio_net_labeller**

Type: `Labeller`.

**FVP_Base_RevC_2xAEMvA.bp.virtio_rng**

VirtioEntropy device over MMIO transport.
Type: `VirtioEntropyMMIO`.

**FVP_Base_RevC_2xAEMvA.bp.virtioblockdevice**

VirtioBlock device over MMIO transport.
Type: `VirtioBlockMMIO`.

**FVP_Base_RevC_2xAEMvA.bp.virtioblockdevice_labeller**

Type: `Labeller`.

**FVP_Base_RevC_2xAEMvA.bp.virtiop9device**

virtio P9 server.
Type: `VirtioP9Device`.

**FVP_Base_RevC_2xAEMvA.bp.virtiop9device_labeller**

Type: `Labeller`.

**FVP_Base_RevC_2xAEMvA.bp.vis**

Display window for VE using Visualisation library.
Type: `VEVisualisation`.

**FVP_Base_RevC_2xAEMvA.bp.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_Base_RevC_2xAEMvA.bp.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_RevC_2xAEMvA.bp.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_RevC_2xAEMvA.bp.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_RevC_2xAEMvA.bp.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_RevC_2xAEMvA.bp.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_RevC_2xAEMvA.bp.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_RevC_2xAEMvA.bp.vram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_Base_RevC_2xAEMvA.cci550**

This is a cache coherent interconnect.
Type: `CCI550`.

**FVP_Base_RevC_2xAEMvA.clockdivider0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_RevC_2xAEMvA.clockdivider1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_Base_RevC_2xAEMvA.cluster0**

ARM AEMvA Cluster CT model.
Type: `Cluster_ARM_AEM-A_MP`.

**FVP_Base_RevC_2xAEMvA.cluster0.cpu0**

ARM AEM-A MP CT model.
Type: `ARM_AEM-A_MP`.

**FVP_Base_RevC_2xAEMvA.cluster0.cpu0.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_RevC_2xAEMvA.cluster0.cpu0.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_RevC_2xAEMvA.cluster0.cpu0.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_RevC_2xAEMvA.cluster0.cpu1**

ARM AEM-A MP CT model.
Type: `ARM_AEM-A_MP`.

**FVP_Base_RevC_2xAEMvA.cluster0.cpu1.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_RevC_2xAEMvA.cluster0.cpu1.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_RevC_2xAEMvA.cluster0.cpu1.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_RevC_2xAEMvA.cluster0.cpu2**

ARM AEM-A MP CT model.
Type: `ARM_AEM-A_MP`.

**FVP_Base_RevC_2xAEMvA.cluster0.cpu2.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_RevC_2xAEMvA.cluster0.cpu2.l1dcache**

PV Cache.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `PVCache`.

**FVP_Base_RevC_2xAEMvA.cluster0.cpu2.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_RevC_2xAEMvA.cluster0.cpu3**

ARM AEM-A MP CT model.
Type: `ARM_AEM-A_MP`.

**FVP_Base_RevC_2xAEMvA.cluster0.cpu3.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_RevC_2xAEMvA.cluster0.cpu3.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_RevC_2xAEMvA.cluster0.cpu3.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_RevC_2xAEMvA.cluster0.l2_cache**

PV Cache.
Type: `PVCache`.

**FVP_Base_RevC_2xAEMvA.cluster0_labeller**

Type: `Labeller`.

**FVP_Base_RevC_2xAEMvA.cluster1**

ARM AEMvA Cluster CT model.
Type: `Cluster_ARM_AEM-A_MP`.

**FVP_Base_RevC_2xAEMvA.cluster1.cpu0**

ARM AEM-A MP CT model.
Type: `ARM_AEM-A_MP`.

**FVP_Base_RevC_2xAEMvA.cluster1.cpu0.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_RevC_2xAEMvA.cluster1.cpu0.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_Base_RevC_2xAEMvA.cluster1.cpu0.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_Base_RevC_2xAEMvA.cluster1.cpu1**

ARM AEM-A MP CT model.
Type: `ARM_AEM-A_MP`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_RevC_2xAEMvA.cluster1.cpu1.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_RevC_2xAEMvA.cluster1.cpu1.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_RevC_2xAEMvA.cluster1.cpu1.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_RevC_2xAEMvA.cluster1.cpu2**
ARM AEM-A MP CT model.
Type: `ARM_AEM-A_MP`.

**FVP_Base_RevC_2xAEMvA.cluster1.cpu2.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_RevC_2xAEMvA.cluster1.cpu2.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_RevC_2xAEMvA.cluster1.cpu2.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_RevC_2xAEMvA.cluster1.cpu3**
ARM AEM-A MP CT model.
Type: `ARM_AEM-A_MP`.

**FVP_Base_RevC_2xAEMvA.cluster1.cpu3.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_Base_RevC_2xAEMvA.cluster1.cpu3.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_Base_RevC_2xAEMvA.cluster1.cpu3.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_Base_RevC_2xAEMvA.cluster1.l2_cache**
PV Cache.
Type: `PVCache`.

**FVP_Base_RevC_2xAEMvA.cluster1_labeller**
Type: `Labeller`.

**FVP_Base_RevC_2xAEMvA.dapmemlogger**
Bus Logger.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `PVBusLogger`.

**FVP_Base_RevC_2xAEMvA.elfloader**

ELF loader component.
Type: `ElfLoader`.

**FVP_Base_RevC_2xAEMvA.gic_distributor**

GIC Interrupt Redistribution Infrastucture component.
Type: `GIC_IRI`.

**FVP_Base_RevC_2xAEMvA.gpu**

G76 addon for the Base Platform.
Type: `G76_DB`.

**FVP_Base_RevC_2xAEMvA.gpu.gpu**

ARM Mali-G76 GPU.
Type: `Mali_G76`.

**FVP_Base_RevC_2xAEMvA.pci**

PCI addon for the Base Platform.
Type: `BasePlatformPCIRevC`.

**FVP_Base_RevC_2xAEMvA.pci.ahci_pci**

Type: `AHCI_PCI`.

**FVP_Base_RevC_2xAEMvA.pci.ahci_pci.ahci**

AHCI controller with attached SATA disks and PCIe interface.
Type: `AHCI_SATA`.

**FVP_Base_RevC_2xAEMvA.pci.ahci_pci.buslogger**

Bus Logger.
Type: `PVBusLogger`.

**FVP_Base_RevC_2xAEMvA.pci.ahci_pci.pcidevice**

PCI Wrapper for memory mapped components.
Type: `PCIDevice`.

**FVP_Base_RevC_2xAEMvA.pci.ahci_pci.pcidevice.dmalogger**

Bus Logger.
Type: `PVBusLogger`.

**FVP_Base_RevC_2xAEMvA.pci.ahci_pci.pcidevice.incoming_memory_logger**

Bus Logger.
Type: `PVBusLogger`.

**FVP_Base_RevC_2xAEMvA.pci.ahci_pci.pcidevice.lost_mastered_transactions**

Bus Logger.
Type: `PVBusLogger`.

**FVP_Base_RevC_2xAEMvA.pci.ahci_pci.pcidevice.lost_transactions_to_pcie**

Bus Logger.
Type: `PVBusLogger`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_RevC_2xAEMvA.pci.ahci_pci.pcidevice.msix_pba_logger**

Bus Logger.
Type: PVBusLogger.

**FVP_Base_RevC_2xAEMvA.pci.ahci_pci.pcidevice.msix_table_logger**

Bus Logger.
Type: PVBusLogger.

**FVP_Base_RevC_2xAEMvA.pci.ahci_pci.pcidevice.to_client_memory_logger**

Bus Logger.
Type: PVBusLogger.

**FVP_Base_RevC_2xAEMvA.pci.pci_smmuv3**

System MMUv3 configured for PCI-E Sub-system.
Type: SMMUv3_FOR_PCIE.

**FVP_Base_RevC_2xAEMvA.pci.pci_smmuv3.mmu**

SMMUv3 AEM.
Type: SMMUv3AEM.

**FVP_Base_RevC_2xAEMvA.pci.pci_smmuv3_msirewriter**

Recognise writes to the GITS_TRANSLATER register and rewrite them to go to the GITS_TRANSLATE64R register. The DeviceID is expected to be in the bottom 32 bits of ExtendedID of the transaction. Debug transactions and reads are not rewritten. This component can also, optionally, apply a 16 bit 'label' to the top 16 bits of the MasterID in the same way that the Labeller component does.
Type: MSIRewriter.

**FVP_Base_RevC_2xAEMvA.pci.pcidevice0**

PCIeEndpoint.
Type: PCIeEndpoint.

**FVP_Base_RevC_2xAEMvA.pci.pcidevice1**

PCIeEndpoint.
Type: PCIeEndpoint.

**FVP_Base_RevC_2xAEMvA.pci.pcivirtioblockdevice0**

virtio PCI block device.
Type: VirtioPCIBlockDevice.

**FVP_Base_RevC_2xAEMvA.pci.pcivirtioblockdevice1**

virtio PCI block device.
Type: VirtioPCIBlockDevice.

**FVP_Base_RevC_2xAEMvA.pci.pvbus2pci**

PVBus2PCI_2.
Type: PVBus2PCI_2.

**FVP_Base_RevC_2xAEMvA.pci.smmulogger**

Bus Logger.
Type: PVBusLogger.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_Base_RevC_2xAEMvA.pci.tbu0_pre_smmu_logger**

Bus Logger.
Type: `PVBusLogger`.

**FVP_Base_RevC_2xAEMvA.pctl**

Base Platforms Power Controller.
Type: `Base_PowerController`.

**FVP_Base_RevC_2xAEMvA.pctl.timer_reset**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_Base_RevC_2xAEMvA.pctl.timer_reset.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_Base_RevC_2xAEMvA.pctl.timer_reset.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_Base_RevC_2xAEMvA.pctl.timer_reset.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_Base_RevC_2xAEMvA.test_engine**

Test Engine for base platform.
Type: `TestEngine`.

**FVP_Base_RevC_2xAEMvA.test_engine.device0**

SMMUv3 Test Engine.
Type: `SMMUv3TestEngine`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

# 14. BaseR Platform FVPs

This chapter describes the BaseR Platform FVPs contained in the FVP Standard Library.

For the BaseR Platform memory map, see BaseR Platform memory map in the *Fast Models Reference Guide*.

## 14.1 FVP_BaseR_Cortex-R52Plus

List of instances in FVP_BaseR_Cortex-R52Plus.

### FVP_BaseR_Cortex-R52Plus instances

**FVP_BaseR_Cortex_R52Plus**

Base Platform Compute Subsystem for ARMCortexR52PlusCT.
Type: `FVP_BaseR_Cortex_R52Plus`.

**FVP_BaseR_Cortex_R52Plus.bp**

Peripherals and address map for the Base Platform.
Type: `BasePlatformPeripherals`.

**FVP_BaseR_Cortex_R52Plus.bp.Timer_0_1**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_BaseR_Cortex_R52Plus.bp.Timer_0_1.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R52Plus.bp.Timer_0_1.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R52Plus.bp.Timer_0_1.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_BaseR_Cortex_R52Plus.bp.Timer_0_1.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_BaseR_Cortex_R52Plus.bp.Timer_2_3**

ARM Dual-Timer Module(SP804).

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `SP804_Timer`.

**FVP_BaseR_Cortex_R52Plus.bp.Timer_2_3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R52Plus.bp.Timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R52Plus.bp.Timer_2_3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_BaseR_Cortex_R52Plus.bp.Timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_BaseR_Cortex_R52Plus.bp.ap_refclk**

ARM Generic Timer.
Type: `MemoryMappedGenericTimer`.

**FVP_BaseR_Cortex_R52Plus.bp.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**FVP_BaseR_Cortex_R52Plus.bp.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R52Plus.bp.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R52Plus.bp.clock300MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_BaseR_Cortex_R52Plus.bp.clock32KHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R52Plus.bp.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R52Plus.bp.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R52Plus.bp.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R52Plus.bp.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R52Plus.bp.dmc**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R52Plus.bp.dmc_phy**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R52Plus.bp.dummy_local_dap_rom**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R52Plus.bp.dummy_ram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_BaseR_Cortex_R52Plus.bp.dummy_usb**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R52Plus.bp.exclusive_monitor**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_BaseR_Cortex_R52Plus.bp.flash0**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_BaseR_Cortex_R52Plus.bp.flash1**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_BaseR_Cortex_R52Plus.bp.flashloader0**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_BaseR_Cortex_R52Plus.bp.flashloader1**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_BaseR_Cortex_R52Plus.bp.generic_watchdog**

ARM Generic Watchdog.
Type: `MemoryMappedGenericWatchdog`.

**FVP_BaseR_Cortex_R52Plus.bp.hdlcd0**

ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).
Type: `PL370_HDLCD`.

**FVP_BaseR_Cortex_R52Plus.bp.hdlcd0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R52Plus.bp.hdlcd0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_BaseR_Cortex_R52Plus.bp.hdlcd0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R52Plus.bp.hdlcd0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R52Plus.bp.hdlcd0_labeller**

Type: `Labeller`.

**FVP_BaseR_Cortex_R52Plus.bp.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_BaseR_Cortex_R52Plus.bp.mmc**

Generic Multimedia Card.
Type: `MMC`.

**FVP_BaseR_Cortex_R52Plus.bp.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R52Plus.bp.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R52Plus.bp.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R52Plus.bp.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R52Plus.bp.mpe**

Encrypt memory transactions for each encryption context with an independent key to prevent mismatch access.
Type: `PVMemoryProtectionEngine`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_BaseR_Cortex_R52Plus.bp.nontrustedrom**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_BaseR_Cortex_R52Plus.bp.nontrustedromloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_BaseR_Cortex_R52Plus.bp.ns_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R52Plus.bp.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_BaseR_Cortex_R52Plus.bp.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R52Plus.bp.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R52Plus.bp.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R52Plus.bp.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R52Plus.bp.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_BaseR_Cortex_R52Plus.bp.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_BaseR_Cortex_R52Plus.bp.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R52Plus.bp.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R52Plus.bp.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R52Plus.bp.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R52Plus.bp.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R52Plus.bp.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_BaseR_Cortex_R52Plus.bp.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_BaseR_Cortex_R52Plus.bp.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R52Plus.bp.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R52Plus.bp.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R52Plus.bp.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R52Plus.bp.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_BaseR_Cortex_R52Plus.bp.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R52Plus.bp.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_BaseR_Cortex_R52Plus.bp.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R52Plus.bp.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R52Plus.bp.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R52Plus.bp.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

**FVP_BaseR_Cortex_R52Plus.bp.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R52Plus.bp.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R52Plus.bp.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R52Plus.bp.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_BaseR_Cortex_R52Plus.bp.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_BaseR_Cortex_R52Plus.bp.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R52Plus.bp.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R52Plus.bp.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R52Plus.bp.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R52Plus.bp.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_BaseR_Cortex_R52Plus.bp.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R52Plus.bp.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_BaseR_Cortex_R52Plus.bp.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R52Plus.bp.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_BaseR_Cortex_R52Plus.bp.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

**FVP_BaseR_Cortex_R52Plus.bp.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R52Plus.bp.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R52Plus.bp.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R52Plus.bp.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R52Plus.bp.pl111_clcd_labeller**

Type: `Labeller`.

**FVP_BaseR_Cortex_R52Plus.bp.pl180_mci**

ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

**FVP_BaseR_Cortex_R52Plus.bp.ps2keyboard**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_BaseR_Cortex_R52Plus.bp.ps2keyboard.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R52Plus.bp.ps2keyboard.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R52Plus.bp.ps2keyboard.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R52Plus.bp.ps2keyboard.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R52Plus.bp.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

**FVP_BaseR_Cortex_R52Plus.bp.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R52Plus.bp.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R52Plus.bp.ps2mouse.ps2_clocktimer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R52Plus.bp.ps2mouse.ps2_clocktimer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R52Plus.bp.psram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R52Plus.bp.refcounter**
Memory Mapped Counter Module for Generic Timers.
Type: `MemoryMappedCounterModule`.

**FVP_BaseR_Cortex_R52Plus.bp.reset_or**
Or Gate.
Type: `OrGate`.

**FVP_BaseR_Cortex_R52Plus.bp.rl_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R52Plus.bp.rt_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R52Plus.bp.s_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R52Plus.bp.secureDRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R52Plus.bp.secureSRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R52Plus.bp.secureflash**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_BaseR_Cortex_R52Plus.bp.secureflashloader**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_BaseR_Cortex_R52Plus.bp.smsc_91c111**

SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_BaseR_Cortex_R52Plus.bp.sp805_wdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_BaseR_Cortex_R52Plus.bp.sp805_wdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R52Plus.bp.sp805_wdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R52Plus.bp.sp805_wdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R52Plus.bp.sp805_wdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R52Plus.bp.sp810_sysctrl**

Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_BaseR_Cortex_R52Plus.bp.sp810_sysctrl.clkdiv_clk0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R52Plus.bp.sp810_sysctrl.clkdiv_clk1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R52Plus.bp.sp810_sysctrl.clkdiv_clk2**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R52Plus.bp.sp810_sysctrl.clkdiv_clk3**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R52Plus.bp.sram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R52Plus.bp.terminal_0**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_BaseR_Cortex_R52Plus.bp.terminal_1**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_BaseR_Cortex_R52Plus.bp.terminal_2**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_BaseR_Cortex_R52Plus.bp.terminal_3**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_BaseR_Cortex_R52Plus.bp.trusted_key_storage**

Trusted Root-Key Storage unit.
Type: `RootKeyStorage`.

**FVP_BaseR_Cortex_R52Plus.bp.trusted_nv_counter**

Trusted Non-Volatile Counter unit.
Type: `NonVolatileCounter`.

**FVP_BaseR_Cortex_R52Plus.bp.trusted_rng**

Random Number Generator unit.
Type: `RandomNumberGenerator`.

**FVP_BaseR_Cortex_R52Plus.bp.trusted_watchdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_BaseR_Cortex_R52Plus.bp.trusted_watchdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R52Plus.bp.trusted_watchdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R52Plus.bp.trusted_watchdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R52Plus.bp.trusted_watchdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R52Plus.bp.tzc_400**

TrustZone Address Space Controller.
Type: `TZC_400`.

**FVP_BaseR_Cortex_R52Plus.bp.ve_sysregs**

Type: `VE_SysRegs`.

**FVP_BaseR_Cortex_R52Plus.bp.vedcc**

Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

**FVP_BaseR_Cortex_R52Plus.bp.virtio_net**

VirtioNet device over MMIO transport.
Type: `VirtioNetMMIO`.

**FVP_BaseR_Cortex_R52Plus.bp.virtio_net_labeller**

Type: `Labeller`.

**FVP_BaseR_Cortex_R52Plus.bp.virtio_rng**

VirtioEntropy device over MMIO transport.
Type: `VirtioEntropyMMIO`.

**FVP_BaseR_Cortex_R52Plus.bp.virtioblockdevice**

VirtioBlock device over MMIO transport.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `VirtioBlockMMIO`.

**FVP_BaseR_Cortex_R52Plus.bp.virtioblockdevice_labeller**

Type: `Labeller`.

**FVP_BaseR_Cortex_R52Plus.bp.virtiop9device**

virtio P9 server.
Type: `VirtioP9Device`.

**FVP_BaseR_Cortex_R52Plus.bp.virtiop9device_labeller**

Type: `Labeller`.

**FVP_BaseR_Cortex_R52Plus.bp.vis**

Display window for VE using Visualisation library.
Type: `VEVisualisation`.

**FVP_BaseR_Cortex_R52Plus.bp.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_BaseR_Cortex_R52Plus.bp.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R52Plus.bp.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R52Plus.bp.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R52Plus.bp.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_BaseR_Cortex_R52Plus.bp.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R52Plus.bp.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R52Plus.bp.vram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R52Plus.cci400**

Cache Coherent Interconnect for AXI4 ACE.
Type: `CCI400`.

**FVP_BaseR_Cortex_R52Plus.clockdivider0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R52Plus.cluster0**

ARM Cortex-R52Plus Cluster CT model.
Type: `Cluster_ARM_Cortex-R52Plus`.

**FVP_BaseR_Cortex_R52Plus.cluster0.cpu0**

ARM Cortex-R52Plus CT model.
Type: `ARM_Cortex-R52Plus`.

**FVP_BaseR_Cortex_R52Plus.cluster0.cpu0.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_BaseR_Cortex_R52Plus.cluster0.cpu0.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_BaseR_Cortex_R52Plus.cluster0.cpu0.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_BaseR_Cortex_R52Plus.cluster0.cpu1**

ARM Cortex-R52Plus CT model.
Type: `ARM_Cortex-R52Plus`.

**FVP_BaseR_Cortex_R52Plus.cluster0.cpu1.dtlb**

TLB - instruction, data or unified.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `TlbCadi`.

**FVP_BaseR_Cortex_R52Plus.cluster0.cpu1.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_BaseR_Cortex_R52Plus.cluster0.cpu1.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_BaseR_Cortex_R52Plus.cluster0.cpu2**

ARM Cortex-R52Plus CT model.
Type: `ARM_Cortex-R52Plus`.

**FVP_BaseR_Cortex_R52Plus.cluster0.cpu2.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_BaseR_Cortex_R52Plus.cluster0.cpu2.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_BaseR_Cortex_R52Plus.cluster0.cpu2.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_BaseR_Cortex_R52Plus.cluster0.cpu3**

ARM Cortex-R52Plus CT model.
Type: `ARM_Cortex-R52Plus`.

**FVP_BaseR_Cortex_R52Plus.cluster0.cpu3.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_BaseR_Cortex_R52Plus.cluster0.cpu3.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_BaseR_Cortex_R52Plus.cluster0.cpu3.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_BaseR_Cortex_R52Plus.cluster0.gic_iri**

GIC IRI internal to cluster.
Type: `gic_iri`.

**FVP_BaseR_Cortex_R52Plus.cluster0_labeller**

Type: `Labeller`.

**FVP_BaseR_Cortex_R52Plus.dapmemlogger**

Bus Logger.
Type: `PVBusLogger`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_BaseR_Cortex_R52Plus.elfloader**

ELF loader component.
Type: `ElfLoader`.

**FVP_BaseR_Cortex_R52Plus.flash_ram0**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R52Plus.pctl**

Base Platforms Power Controller.
Type: `Base_PowerController`.

**FVP_BaseR_Cortex_R52Plus.pctl.timer_reset**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R52Plus.pctl.timer_reset.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R52Plus.pctl.timer_reset.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R52Plus.pctl.timer_reset.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

## 14.2 FVP_BaseR_Cortex-R52x1

List of instances in FVP_BaseR_Cortex-R52x1.

### FVP_BaseR_Cortex-R52x1 instances

**FVP_BaseR_Cortex_R52x1**

Base Platform Compute Subsystem for ARMCortexR52x1CT.
Type: `FVP_BaseR_Cortex_R52x1`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_BaseR_Cortex_R52x1.bp**

Peripherals and address map for the Base Platform.
Type: `BasePlatformPeripherals`.

**FVP_BaseR_Cortex_R52x1.bp.Timer_0_1**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_BaseR_Cortex_R52x1.bp.Timer_0_1.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R52x1.bp.Timer_0_1.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R52x1.bp.Timer_0_1.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_BaseR_Cortex_R52x1.bp.Timer_0_1.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_BaseR_Cortex_R52x1.bp.Timer_2_3**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_BaseR_Cortex_R52x1.bp.Timer_2_3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R52x1.bp.Timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R52x1.bp.Timer_2_3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_BaseR_Cortex_R52x1.bp.Timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_BaseR_Cortex_R52x1.bp.ap_refclk**

ARM Generic Timer.
Type: `MemoryMappedGenericTimer`.

**FVP_BaseR_Cortex_R52x1.bp.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**FVP_BaseR_Cortex_R52x1.bp.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R52x1.bp.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R52x1.bp.clock300MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R52x1.bp.clock32KHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R52x1.bp.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R52x1.bp.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R52x1.bp.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R52x1.bp.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R52x1.bp.dmc**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R52x1.bp.dmc_phy**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R52x1.bp.dummy_local_dap_rom**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R52x1.bp.dummy_ram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R52x1.bp.dummy_usb**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R52x1.bp.exclusive_monitor**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_BaseR_Cortex_R52x1.bp.flash0**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_BaseR_Cortex_R52x1.bp.flash1**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_BaseR_Cortex_R52x1.bp.flashloader0**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_BaseR_Cortex_R52x1.bp.flashloader1**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_BaseR_Cortex_R52x1.bp.generic_watchdog**

ARM Generic Watchdog.
Type: `MemoryMappedGenericWatchdog`.

**FVP_BaseR_Cortex_R52x1.bp.hdlcd0**

ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).
Type: `PL370_HDLCD`.

**FVP_BaseR_Cortex_R52x1.bp.hdlcd0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R52x1.bp.hdlcd0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R52x1.bp.hdlcd0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R52x1.bp.hdlcd0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R52x1.bp.hdlcd0_labeller**

Type: `Labeller`.

**FVP_BaseR_Cortex_R52x1.bp.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_BaseR_Cortex_R52x1.bp.mmc**

Generic Multimedia Card.
Type: `MMC`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_BaseR_Cortex_R52x1.bp.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R52x1.bp.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R52x1.bp.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R52x1.bp.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R52x1.bp.mpe**

Encrypt memory transactions for each encryption context with an independent key to prevent mismatch access.
Type: `PVMemoryProtectionEngine`.

**FVP_BaseR_Cortex_R52x1.bp.nontrustedrom**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_BaseR_Cortex_R52x1.bp.nontrustedromloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_BaseR_Cortex_R52x1.bp.ns_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R52x1.bp.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_BaseR_Cortex_R52x1.bp.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R52x1.bp.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R52x1.bp.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R52x1.bp.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R52x1.bp.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R52x1.bp.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_BaseR_Cortex_R52x1.bp.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R52x1.bp.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R52x1.bp.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R52x1.bp.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R52x1.bp.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R52x1.bp.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_BaseR_Cortex_R52x1.bp.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R52x1.bp.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R52x1.bp.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_BaseR_Cortex_R52x1.bp.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R52x1.bp.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R52x1.bp.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_BaseR_Cortex_R52x1.bp.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R52x1.bp.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R52x1.bp.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R52x1.bp.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R52x1.bp.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R52x1.bp.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_BaseR_Cortex_R52x1.bp.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R52x1.bp.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R52x1.bp.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R52x1.bp.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R52x1.bp.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_BaseR_Cortex_R52x1.bp.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R52x1.bp.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_BaseR_Cortex_R52x1.bp.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R52x1.bp.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R52x1.bp.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_BaseR_Cortex_R52x1.bp.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R52x1.bp.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_BaseR_Cortex_R52x1.bp.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R52x1.bp.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_BaseR_Cortex_R52x1.bp.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

**FVP_BaseR_Cortex_R52x1.bp.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R52x1.bp.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R52x1.bp.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R52x1.bp.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R52x1.bp.pl111_clcd_labeller**

Type: `Labeller`.

**FVP_BaseR_Cortex_R52x1.bp.pl180_mci**

ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

**FVP_BaseR_Cortex_R52x1.bp.ps2keyboard**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

**FVP_BaseR_Cortex_R52x1.bp.ps2keyboard.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R52x1.bp.ps2keyboard.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R52x1.bp.ps2keyboard.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_BaseR_Cortex_R52x1.bp.ps2keyboard.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R52x1.bp.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

**FVP_BaseR_Cortex_R52x1.bp.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R52x1.bp.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R52x1.bp.ps2mouse.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R52x1.bp.ps2mouse.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R52x1.bp.psram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R52x1.bp.refcounter**

Memory Mapped Counter Module for Generic Timers.
Type: `MemoryMappedCounterModule`.

**FVP_BaseR_Cortex_R52x1.bp.reset_or**

Or Gate.
Type: `OrGate`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_BaseR_Cortex_R52x1.bp.rl_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R52x1.bp.rt_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R52x1.bp.s_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R52x1.bp.secureDRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R52x1.bp.secureSRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R52x1.bp.secureflash**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_BaseR_Cortex_R52x1.bp.secureflashloader**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_BaseR_Cortex_R52x1.bp.smsc_91c111**
SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_BaseR_Cortex_R52x1.bp.sp805_wdog**
ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_BaseR_Cortex_R52x1.bp.sp805_wdog.clocktimer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R52x1.bp.sp805_wdog.clocktimer.timer**
A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R52x1.bp.sp805_wdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R52x1.bp.sp805_wdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R52x1.bp.sp810_sysctrl**

Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_BaseR_Cortex_R52x1.bp.sp810_sysctrl.clkdiv_clk0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R52x1.bp.sp810_sysctrl.clkdiv_clk1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R52x1.bp.sp810_sysctrl.clkdiv_clk2**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R52x1.bp.sp810_sysctrl.clkdiv_clk3**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R52x1.bp.sram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R52x1.bp.terminal_0**

Telnet terminal interface.
Type: `TelnetTerminal`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_BaseR_Cortex_R52x1.bp.terminal_1**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_BaseR_Cortex_R52x1.bp.terminal_2**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_BaseR_Cortex_R52x1.bp.terminal_3**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_BaseR_Cortex_R52x1.bp.trusted_key_storage**

Trusted Root-Key Storage unit.
Type: `RootKeyStorage`.

**FVP_BaseR_Cortex_R52x1.bp.trusted_nv_counter**

Trusted Non-Volatile Counter unit.
Type: `NonVolatileCounter`.

**FVP_BaseR_Cortex_R52x1.bp.trusted_rng**

Random Number Generator unit.
Type: `RandomNumberGenerator`.

**FVP_BaseR_Cortex_R52x1.bp.trusted_watchdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_BaseR_Cortex_R52x1.bp.trusted_watchdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R52x1.bp.trusted_watchdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R52x1.bp.trusted_watchdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_BaseR_Cortex_R52x1.bp.trusted_watchdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R52x1.bp.tzc_400**

TrustZone Address Space Controller.
Type: `TZC_400`.

**FVP_BaseR_Cortex_R52x1.bp.ve_sysregs**

Type: `VE_SysRegs`.

**FVP_BaseR_Cortex_R52x1.bp.vedcc**

Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

**FVP_BaseR_Cortex_R52x1.bp.virtio_net**

VirtioNet device over MMIO transport.
Type: `VirtioNetMMIO`.

**FVP_BaseR_Cortex_R52x1.bp.virtio_net_labeller**

Type: `Labeller`.

**FVP_BaseR_Cortex_R52x1.bp.virtio_rng**

VirtioEntropy device over MMIO transport.
Type: `VirtioEntropyMMIO`.

**FVP_BaseR_Cortex_R52x1.bp.virtioblockdevice**

VirtioBlock device over MMIO transport.
Type: `VirtioBlockMMIO`.

**FVP_BaseR_Cortex_R52x1.bp.virtioblockdevice_labeller**

Type: `Labeller`.

**FVP_BaseR_Cortex_R52x1.bp.virtiop9device**

virtio P9 server.
Type: `VirtioP9Device`.

**FVP_BaseR_Cortex_R52x1.bp.virtiop9device_labeller**

Type: `Labeller`.

**FVP_BaseR_Cortex_R52x1.bp.vis**

Display window for VE using Visualisation library.
Type: `VEVisualisation`.

**FVP_BaseR_Cortex_R52x1.bp.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_BaseR_Cortex_R52x1.bp.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R52x1.bp.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R52x1.bp.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R52x1.bp.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R52x1.bp.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R52x1.bp.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R52x1.bp.vram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R52x1.cci400**

Cache Coherent Interconnect for AXI4 ACE.
Type: `CCI400`.

**FVP_BaseR_Cortex_R52x1.clockdivider0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R52x1.cluster0**
ARM CortexR52 MP Cluster CT model.
Type: `Cluster_ARM_Cortex-R52`.

**FVP_BaseR_Cortex_R52x1.cluster0.cpu0**
ARM CortexR52 MP CT model.
Type: `ARM_CortexR52`.

**FVP_BaseR_Cortex_R52x1.cluster0.cpu0.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_BaseR_Cortex_R52x1.cluster0.cpu0.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_BaseR_Cortex_R52x1.cluster0.cpu0.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_BaseR_Cortex_R52x1.cluster0.gic_iri**
GIC IRI internal to cluster.
Type: `gic_iri`.

**FVP_BaseR_Cortex_R52x1.cluster0_labeller**
Type: `Labeller`.

**FVP_BaseR_Cortex_R52x1.dapmemlogger**
Bus Logger.
Type: `PVBusLogger`.

**FVP_BaseR_Cortex_R52x1.elfloader**
ELF loader component.
Type: `ElfLoader`.

**FVP_BaseR_Cortex_R52x1.flash_ram0**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R52x1.pctl**
Base Platforms Power Controller.
Type: `Base_PowerController`.

**FVP_BaseR_Cortex_R52x1.pctl.timer_reset**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**`FVP_BaseR_Cortex_R52x1.pctl.timer_reset.timer`**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**`FVP_BaseR_Cortex_R52x1.pctl.timer_reset.timer.thread`**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**`FVP_BaseR_Cortex_R52x1.pctl.timer_reset.timer.thread_event`**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

## 14.3 FVP_BaseR_Cortex-R52x2

List of instances in FVP_BaseR_Cortex-R52x2.

### FVP_BaseR_Cortex-R52x2 instances

**`FVP_BaseR_Cortex_R52x2`**

Base Platform Compute Subsystem for ARMCortexR52x2CT.
Type: `FVP_BaseR_Cortex_R52x2`.

**`FVP_BaseR_Cortex_R52x2.bp`**

Peripherals and address map for the Base Platform.
Type: `BasePlatformPeripherals`.

**`FVP_BaseR_Cortex_R52x2.bp.Timer_0_1`**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**`FVP_BaseR_Cortex_R52x2.bp.Timer_0_1.clk_div0`**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**`FVP_BaseR_Cortex_R52x2.bp.Timer_0_1.clk_div1`**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R52x2.bp.Timer_0_1.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_BaseR_Cortex_R52x2.bp.Timer_0_1.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_BaseR_Cortex_R52x2.bp.Timer_2_3**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_BaseR_Cortex_R52x2.bp.Timer_2_3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R52x2.bp.Timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R52x2.bp.Timer_2_3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_BaseR_Cortex_R52x2.bp.Timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_BaseR_Cortex_R52x2.bp.ap_refclk**

ARM Generic Timer.
Type: `MemoryMappedGenericTimer`.

**FVP_BaseR_Cortex_R52x2.bp.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**FVP_BaseR_Cortex_R52x2.bp.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_BaseR_Cortex_R52x2.bp.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R52x2.bp.clock300MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R52x2.bp.clock32KHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R52x2.bp.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R52x2.bp.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R52x2.bp.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R52x2.bp.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R52x2.bp.dmc**

RAM device, can be dynamic or static ram.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `RAMDevice`.

**FVP_BaseR_Cortex_R52x2.bp.dmc_phy**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R52x2.bp.dummy_local_dap_rom**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R52x2.bp.dummy_ram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R52x2.bp.dummy_usb**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R52x2.bp.exclusive_monitor**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_BaseR_Cortex_R52x2.bp.flash0**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_BaseR_Cortex_R52x2.bp.flash1**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_BaseR_Cortex_R52x2.bp.flashloader0**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_BaseR_Cortex_R52x2.bp.flashloader1**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_BaseR_Cortex_R52x2.bp.generic_watchdog**

ARM Generic Watchdog.
Type: `MemoryMappedGenericWatchdog`.

**FVP_BaseR_Cortex_R52x2.bp.hdlcd0**

ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).
Type: `PL370_HDLCD`.

**FVP_BaseR_Cortex_R52x2.bp.hdlcd0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R52x2.bp.hdlcd0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R52x2.bp.hdlcd0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R52x2.bp.hdlcd0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R52x2.bp.hdlcd0_labeller**

Type: `Labeller`.

**FVP_BaseR_Cortex_R52x2.bp.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_BaseR_Cortex_R52x2.bp.mmc**

Generic Multimedia Card.
Type: `MMC`.

**FVP_BaseR_Cortex_R52x2.bp.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R52x2.bp.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_BaseR_Cortex_R52x2.bp.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R52x2.bp.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R52x2.bp.mpe**

Encrypt memory transactions for each encryption context with an independent key to prevent mismatch access.
Type: `PVMemoryProtectionEngine`.

**FVP_BaseR_Cortex_R52x2.bp.nontrustedrom**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_BaseR_Cortex_R52x2.bp.nontrustedromloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_BaseR_Cortex_R52x2.bp.ns_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R52x2.bp.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_BaseR_Cortex_R52x2.bp.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R52x2.bp.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R52x2.bp.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R52x2.bp.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R52x2.bp.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R52x2.bp.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_BaseR_Cortex_R52x2.bp.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R52x2.bp.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R52x2.bp.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R52x2.bp.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R52x2.bp.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_BaseR_Cortex_R52x2.bp.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_BaseR_Cortex_R52x2.bp.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R52x2.bp.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R52x2.bp.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R52x2.bp.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R52x2.bp.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R52x2.bp.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_BaseR_Cortex_R52x2.bp.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_BaseR_Cortex_R52x2.bp.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R52x2.bp.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R52x2.bp.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R52x2.bp.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R52x2.bp.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

**FVP_BaseR_Cortex_R52x2.bp.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R52x2.bp.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_BaseR_Cortex_R52x2.bp.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R52x2.bp.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R52x2.bp.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_BaseR_Cortex_R52x2.bp.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R52x2.bp.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R52x2.bp.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R52x2.bp.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R52x2.bp.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_BaseR_Cortex_R52x2.bp.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_BaseR_Cortex_R52x2.bp.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_BaseR_Cortex_R52x2.bp.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R52x2.bp.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_BaseR_Cortex_R52x2.bp.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

**FVP_BaseR_Cortex_R52x2.bp.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R52x2.bp.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R52x2.bp.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R52x2.bp.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R52x2.bp.pl111_clcd_labeller**

Type: `Labeller`.

**FVP_BaseR_Cortex_R52x2.bp.pl180_mci**

ARM PrimeCell Multimedia Card Interface (PL180).

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `PL180_MCI`.

**FVP_BaseR_Cortex_R52x2.bp.ps2keyboard**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

**FVP_BaseR_Cortex_R52x2.bp.ps2keyboard.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R52x2.bp.ps2keyboard.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R52x2.bp.ps2keyboard.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R52x2.bp.ps2keyboard.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R52x2.bp.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

**FVP_BaseR_Cortex_R52x2.bp.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_BaseR_Cortex_R52x2.bp.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R52x2.bp.ps2mouse.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R52x2.bp.ps2mouse.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R52x2.bp.psram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R52x2.bp.refcounter**

Memory Mapped Counter Module for Generic Timers.
Type: `MemoryMappedCounterModule`.

**FVP_BaseR_Cortex_R52x2.bp.reset_or**

Or Gate.
Type: `OrGate`.

**FVP_BaseR_Cortex_R52x2.bp.rl_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R52x2.bp.rt_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R52x2.bp.s_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R52x2.bp.secureDRAM**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R52x2.bp.secureSRAM**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R52x2.bp.secureflash**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_BaseR_Cortex_R52x2.bp.secureflashloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_BaseR_Cortex_R52x2.bp.smsc_91c111**

SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_BaseR_Cortex_R52x2.bp.sp805_wdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_BaseR_Cortex_R52x2.bp.sp805_wdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R52x2.bp.sp805_wdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R52x2.bp.sp805_wdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R52x2.bp.sp805_wdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R52x2.bp.sp810_sysctrl**

Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_BaseR_Cortex_R52x2.bp.sp810_sysctrl.clkdiv_clk0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_BaseR_Cortex_R52x2.bp.sp810_sysctrl.clkdiv_clk1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R52x2.bp.sp810_sysctrl.clkdiv_clk2**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R52x2.bp.sp810_sysctrl.clkdiv_clk3**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R52x2.bp.sram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R52x2.bp.terminal_0**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_BaseR_Cortex_R52x2.bp.terminal_1**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_BaseR_Cortex_R52x2.bp.terminal_2**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_BaseR_Cortex_R52x2.bp.terminal_3**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_BaseR_Cortex_R52x2.bp.trusted_key_storage**

Trusted Root-Key Storage unit.
Type: `RootKeyStorage`.

**FVP_BaseR_Cortex_R52x2.bp.trusted_nv_counter**

Trusted Non-Volatile Counter unit.
Type: `NonVolatileCounter`.

**FVP_BaseR_Cortex_R52x2.bp.trusted_rng**

Random Number Generator unit.
Type: `RandomNumberGenerator`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_BaseR_Cortex_R52x2.bp.trusted_watchdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_BaseR_Cortex_R52x2.bp.trusted_watchdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R52x2.bp.trusted_watchdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R52x2.bp.trusted_watchdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R52x2.bp.trusted_watchdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R52x2.bp.tzc_400**

TrustZone Address Space Controller.
Type: `TZC_400`.

**FVP_BaseR_Cortex_R52x2.bp.ve_sysregs**

Type: `VE_SysRegs`.

**FVP_BaseR_Cortex_R52x2.bp.vedcc**

Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

**FVP_BaseR_Cortex_R52x2.bp.virtio_net**

VirtioNet device over MMIO transport.
Type: `VirtioNetMMIO`.

**FVP_BaseR_Cortex_R52x2.bp.virtio_net_labeller**

Type: `Labeller`.

**FVP_BaseR_Cortex_R52x2.bp.virtio_rng**

VirtioEntropy device over MMIO transport.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `VirtioEntropyMMIO`.

**FVP_BaseR_Cortex_R52x2.bp.virtioblockdevice**

VirtioBlock device over MMIO transport.
Type: `VirtioBlockMMIO`.

**FVP_BaseR_Cortex_R52x2.bp.virtioblockdevice_labeller**

Type: `Labeller`.

**FVP_BaseR_Cortex_R52x2.bp.virtiop9device**

virtio P9 server.
Type: `VirtioP9Device`.

**FVP_BaseR_Cortex_R52x2.bp.virtiop9device_labeller**

Type: `Labeller`.

**FVP_BaseR_Cortex_R52x2.bp.vis**

Display window for VE using Visualisation library.
Type: `VEVisualisation`.

**FVP_BaseR_Cortex_R52x2.bp.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_BaseR_Cortex_R52x2.bp.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R52x2.bp.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R52x2.bp.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_BaseR_Cortex_R52x2.bp.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R52x2.bp.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R52x2.bp.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R52x2.bp.vram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R52x2.cci400**

Cache Coherent Interconnect for AXI4 ACE.
Type: `CCI400`.

**FVP_BaseR_Cortex_R52x2.clockdivider0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R52x2.cluster0**

ARM CortexR52 MP Cluster CT model.
Type: `Cluster_ARM_Cortex-R52`.

**FVP_BaseR_Cortex_R52x2.cluster0.cpu0**

ARM CortexR52 MP CT model.
Type: `ARM_CortexR52`.

**FVP_BaseR_Cortex_R52x2.cluster0.cpu0.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_BaseR_Cortex_R52x2.cluster0.cpu0.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_BaseR_Cortex_R52x2.cluster0.cpu0.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_BaseR_Cortex_R52x2.cluster0.cpu1**

ARM CortexR52 MP CT model.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ARM_CortexR52`.

**FVP_BaseR_Cortex_R52x2.cluster0.cpu1.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_BaseR_Cortex_R52x2.cluster0.cpu1.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_BaseR_Cortex_R52x2.cluster0.cpu1.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_BaseR_Cortex_R52x2.cluster0.gic_iri**

GIC IRI internal to cluster.
Type: `gic_iri`.

**FVP_BaseR_Cortex_R52x2.cluster0_labeller**

Type: `Labeller`.

**FVP_BaseR_Cortex_R52x2.dapmemlogger**

Bus Logger.
Type: `PVBusLogger`.

**FVP_BaseR_Cortex_R52x2.elfloader**

ELF loader component.
Type: `ElfLoader`.

**FVP_BaseR_Cortex_R52x2.flash_ram0**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R52x2.pctl**

Base Platforms Power Controller.
Type: `Base_PowerController`.

**FVP_BaseR_Cortex_R52x2.pctl.timer_reset**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R52x2.pctl.timer_reset.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**`FVP_BaseR_Cortex_R52x2.pctl.timer_reset.timer.thread`**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**`FVP_BaseR_Cortex_R52x2.pctl.timer_reset.timer.thread_event`**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

## 14.4 FVP_BaseR_Cortex-R52x4

List of instances in FVP_BaseR_Cortex-R52x4.

### FVP_BaseR_Cortex-R52x4 instances

**`FVP_BaseR_Cortex_R52x4`**
Base Platform Compute Subsystem for ARMCortexR52x4CT.
Type: `FVP_BaseR_Cortex_R52x4`.

**`FVP_BaseR_Cortex_R52x4.bp`**
Peripherals and address map for the Base Platform.
Type: `BasePlatformPeripherals`.

**`FVP_BaseR_Cortex_R52x4.bp.Timer_0_1`**
ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**`FVP_BaseR_Cortex_R52x4.bp.Timer_0_1.clk_div0`**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**`FVP_BaseR_Cortex_R52x4.bp.Timer_0_1.clk_div1`**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**`FVP_BaseR_Cortex_R52x4.bp.Timer_0_1.counter0`**
Internal component used by SP804 Timer module.
Type: `CounterModule`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_BaseR_Cortex_R52x4.bp.Timer_0_1.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_BaseR_Cortex_R52x4.bp.Timer_2_3**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_BaseR_Cortex_R52x4.bp.Timer_2_3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R52x4.bp.Timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R52x4.bp.Timer_2_3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_BaseR_Cortex_R52x4.bp.Timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_BaseR_Cortex_R52x4.bp.ap_refclk**

ARM Generic Timer.
Type: `MemoryMappedGenericTimer`.

**FVP_BaseR_Cortex_R52x4.bp.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**FVP_BaseR_Cortex_R52x4.bp.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R52x4.bp.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_BaseR_Cortex_R52x4.bp.clock300MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R52x4.bp.clock32KHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R52x4.bp.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R52x4.bp.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R52x4.bp.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R52x4.bp.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R52x4.bp.dmc**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R52x4.bp.dmc_phy**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_BaseR_Cortex_R52x4.bp.dummy_local_dap_rom**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R52x4.bp.dummy_ram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R52x4.bp.dummy_usb**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R52x4.bp.exclusive_monitor**
Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_BaseR_Cortex_R52x4.bp.flash0**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_BaseR_Cortex_R52x4.bp.flash1**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_BaseR_Cortex_R52x4.bp.flashloader0**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_BaseR_Cortex_R52x4.bp.flashloader1**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_BaseR_Cortex_R52x4.bp.generic_watchdog**
ARM Generic Watchdog.
Type: `MemoryMappedGenericWatchdog`.

**FVP_BaseR_Cortex_R52x4.bp.hdlcd0**
ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).
Type: `PL370_HDLCD`.

**FVP_BaseR_Cortex_R52x4.bp.hdlcd0.timer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R52x4.bp.hdlcd0.timer.timer**
A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R52x4.bp.hdlcd0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R52x4.bp.hdlcd0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R52x4.bp.hdlcd0_labeller**

Type: `Labeller`.

**FVP_BaseR_Cortex_R52x4.bp.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_BaseR_Cortex_R52x4.bp.mmc**

Generic Multimedia Card.
Type: `MMC`.

**FVP_BaseR_Cortex_R52x4.bp.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R52x4.bp.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R52x4.bp.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R52x4.bp.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R52x4.bp.mpe**

Encrypt memory transactions for each encryption context with an independent key to prevent mismatch access.
Type: `PVMemoryProtectionEngine`.

**FVP_BaseR_Cortex_R52x4.bp.nontrustedrom**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_BaseR_Cortex_R52x4.bp.nontrustedromloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_BaseR_Cortex_R52x4.bp.ns_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R52x4.bp.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_BaseR_Cortex_R52x4.bp.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R52x4.bp.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R52x4.bp.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R52x4.bp.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_BaseR_Cortex_R52x4.bp.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R52x4.bp.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_BaseR_Cortex_R52x4.bp.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R52x4.bp.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R52x4.bp.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R52x4.bp.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R52x4.bp.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R52x4.bp.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_BaseR_Cortex_R52x4.bp.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R52x4.bp.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R52x4.bp.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R52x4.bp.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R52x4.bp.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R52x4.bp.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_BaseR_Cortex_R52x4.bp.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R52x4.bp.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R52x4.bp.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R52x4.bp.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R52x4.bp.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R52x4.bp.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

**FVP_BaseR_Cortex_R52x4.bp.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R52x4.bp.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R52x4.bp.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R52x4.bp.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_BaseR_Cortex_R52x4.bp.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_BaseR_Cortex_R52x4.bp.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R52x4.bp.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R52x4.bp.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R52x4.bp.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R52x4.bp.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_BaseR_Cortex_R52x4.bp.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R52x4.bp.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_BaseR_Cortex_R52x4.bp.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R52x4.bp.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_BaseR_Cortex_R52x4.bp.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

**FVP_BaseR_Cortex_R52x4.bp.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R52x4.bp.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R52x4.bp.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R52x4.bp.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R52x4.bp.pl111_clcd_labeller**

Type: `Labeller`.

**FVP_BaseR_Cortex_R52x4.bp.pl180_mci**

ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

**FVP_BaseR_Cortex_R52x4.bp.ps2keyboard**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_BaseR_Cortex_R52x4.bp.ps2keyboard.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R52x4.bp.ps2keyboard.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R52x4.bp.ps2keyboard.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R52x4.bp.ps2keyboard.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R52x4.bp.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

**FVP_BaseR_Cortex_R52x4.bp.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R52x4.bp.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R52x4.bp.ps2mouse.ps2_clocktimer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R52x4.bp.ps2mouse.ps2_clocktimer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R52x4.bp.psram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R52x4.bp.refcounter**
Memory Mapped Counter Module for Generic Timers.
Type: `MemoryMappedCounterModule`.

**FVP_BaseR_Cortex_R52x4.bp.reset_or**
Or Gate.
Type: `OrGate`.

**FVP_BaseR_Cortex_R52x4.bp.rl_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R52x4.bp.rt_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R52x4.bp.s_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R52x4.bp.secureDRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R52x4.bp.secureSRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R52x4.bp.secureflash**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_BaseR_Cortex_R52x4.bp.secureflashloader**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_BaseR_Cortex_R52x4.bp.smsc_91c111**

SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_BaseR_Cortex_R52x4.bp.sp805_wdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_BaseR_Cortex_R52x4.bp.sp805_wdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R52x4.bp.sp805_wdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R52x4.bp.sp805_wdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R52x4.bp.sp805_wdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R52x4.bp.sp810_sysctrl**

Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_BaseR_Cortex_R52x4.bp.sp810_sysctrl.clkdiv_clk0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R52x4.bp.sp810_sysctrl.clkdiv_clk1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R52x4.bp.sp810_sysctrl.clkdiv_clk2**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R52x4.bp.sp810_sysctrl.clkdiv_clk3**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R52x4.bp.sram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R52x4.bp.terminal_0**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_BaseR_Cortex_R52x4.bp.terminal_1**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_BaseR_Cortex_R52x4.bp.terminal_2**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_BaseR_Cortex_R52x4.bp.terminal_3**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_BaseR_Cortex_R52x4.bp.trusted_key_storage**

Trusted Root-Key Storage unit.
Type: `RootKeyStorage`.

**FVP_BaseR_Cortex_R52x4.bp.trusted_nv_counter**

Trusted Non-Volatile Counter unit.
Type: `NonVolatileCounter`.

**FVP_BaseR_Cortex_R52x4.bp.trusted_rng**

Random Number Generator unit.
Type: `RandomNumberGenerator`.

**FVP_BaseR_Cortex_R52x4.bp.trusted_watchdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_BaseR_Cortex_R52x4.bp.trusted_watchdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R52x4.bp.trusted_watchdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R52x4.bp.trusted_watchdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R52x4.bp.trusted_watchdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R52x4.bp.tzc_400**

TrustZone Address Space Controller.
Type: `TZC_400`.

**FVP_BaseR_Cortex_R52x4.bp.ve_sysregs**

Type: `VE_SysRegs`.

**FVP_BaseR_Cortex_R52x4.bp.vedcc**

Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

**FVP_BaseR_Cortex_R52x4.bp.virtio_net**

VirtioNet device over MMIO transport.
Type: `VirtioNetMMIO`.

**FVP_BaseR_Cortex_R52x4.bp.virtio_net_labeller**

Type: `Labeller`.

**FVP_BaseR_Cortex_R52x4.bp.virtio_rng**

VirtioEntropy device over MMIO transport.
Type: `VirtioEntropyMMIO`.

**FVP_BaseR_Cortex_R52x4.bp.virtioblockdevice**

VirtioBlock device over MMIO transport.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `VirtioBlockMMIO`.

**FVP_BaseR_Cortex_R52x4.bp.virtioblockdevice_labeller**

Type: `Labeller`.

**FVP_BaseR_Cortex_R52x4.bp.virtiop9device**

virtio P9 server.
Type: `VirtioP9Device`.

**FVP_BaseR_Cortex_R52x4.bp.virtiop9device_labeller**

Type: `Labeller`.

**FVP_BaseR_Cortex_R52x4.bp.vis**

Display window for VE using Visualisation library.
Type: `VEVisualisation`.

**FVP_BaseR_Cortex_R52x4.bp.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_BaseR_Cortex_R52x4.bp.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R52x4.bp.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R52x4.bp.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R52x4.bp.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_BaseR_Cortex_R52x4.bp.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R52x4.bp.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R52x4.bp.vram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R52x4.cci400**

Cache Coherent Interconnect for AXI4 ACE.
Type: `CCI400`.

**FVP_BaseR_Cortex_R52x4.clockdivider0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R52x4.cluster0**

ARM CortexR52 MP Cluster CT model.
Type: `Cluster_ARM_Cortex-R52`.

**FVP_BaseR_Cortex_R52x4.cluster0.cpu0**

ARM CortexR52 MP CT model.
Type: `ARM_CortexR52`.

**FVP_BaseR_Cortex_R52x4.cluster0.cpu0.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_BaseR_Cortex_R52x4.cluster0.cpu0.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_BaseR_Cortex_R52x4.cluster0.cpu0.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_BaseR_Cortex_R52x4.cluster0.cpu1**

ARM CortexR52 MP CT model.
Type: `ARM_CortexR52`.

**FVP_BaseR_Cortex_R52x4.cluster0.cpu1.dtlb**

TLB - instruction, data or unified.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `TlbCadi`.

**FVP_BaseR_Cortex_R52x4.cluster0.cpu1.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_BaseR_Cortex_R52x4.cluster0.cpu1.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_BaseR_Cortex_R52x4.cluster0.cpu2**

ARM CortexR52 MP CT model.
Type: `ARM_CortexR52`.

**FVP_BaseR_Cortex_R52x4.cluster0.cpu2.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_BaseR_Cortex_R52x4.cluster0.cpu2.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_BaseR_Cortex_R52x4.cluster0.cpu2.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_BaseR_Cortex_R52x4.cluster0.cpu3**

ARM CortexR52 MP CT model.
Type: `ARM_CortexR52`.

**FVP_BaseR_Cortex_R52x4.cluster0.cpu3.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_BaseR_Cortex_R52x4.cluster0.cpu3.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_BaseR_Cortex_R52x4.cluster0.cpu3.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_BaseR_Cortex_R52x4.cluster0.gic_iri**

GIC IRI internal to cluster.
Type: `gic_iri`.

**FVP_BaseR_Cortex_R52x4.cluster0_labeller**

Type: `Labeller`.

**FVP_BaseR_Cortex_R52x4.dapmemlogger**

Bus Logger.
Type: `PVBusLogger`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_BaseR_Cortex_R52x4.elfloader**

ELF loader component.
Type: `ElfLoader`.

**FVP_BaseR_Cortex_R52x4.flash_ram0**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R52x4.pctl**

Base Platforms Power Controller.
Type: `Base_PowerController`.

**FVP_BaseR_Cortex_R52x4.pctl.timer_reset**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R52x4.pctl.timer_reset.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R52x4.pctl.timer_reset.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R52x4.pctl.timer_reset.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

# 14.5 FVP_BaseR_Cortex-R82R1

List of instances in FVP_BaseR_Cortex-R82R1.

## FVP_BaseR_Cortex-R82R1 instances

**FVP_BaseR_Cortex_R82R1**

Base Platform Compute Subsystem for ARMCortexR82CT.
Type: `FVP_BaseR_Cortex_R82R1`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_BaseR_Cortex_R82R1.bp**

Peripherals and address map for the Base Platform.
Type: `BasePlatformPeripherals`.

**FVP_BaseR_Cortex_R82R1.bp.Timer_0_1**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_BaseR_Cortex_R82R1.bp.Timer_0_1.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R82R1.bp.Timer_0_1.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R82R1.bp.Timer_0_1.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_BaseR_Cortex_R82R1.bp.Timer_0_1.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_BaseR_Cortex_R82R1.bp.Timer_2_3**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_BaseR_Cortex_R82R1.bp.Timer_2_3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R82R1.bp.Timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R82R1.bp.Timer_2_3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_BaseR_Cortex_R82R1.bp.Timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_BaseR_Cortex_R82R1.bp.ap_refclk**

ARM Generic Timer.
Type: `MemoryMappedGenericTimer`.

**FVP_BaseR_Cortex_R82R1.bp.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**FVP_BaseR_Cortex_R82R1.bp.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R82R1.bp.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R82R1.bp.clock300MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R82R1.bp.clock32KHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R82R1.bp.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R82R1.bp.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R82R1.bp.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R82R1.bp.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R82R1.bp.dmc**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R82R1.bp.dmc_phy**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R82R1.bp.dummy_local_dap_rom**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R82R1.bp.dummy_ram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R82R1.bp.dummy_usb**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R82R1.bp.exclusive_monitor**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_BaseR_Cortex_R82R1.bp.flash0**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_BaseR_Cortex_R82R1.bp.flash1**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_BaseR_Cortex_R82R1.bp.flashloader0**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_BaseR_Cortex_R82R1.bp.flashloader1**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_BaseR_Cortex_R82R1.bp.generic_watchdog**

ARM Generic Watchdog.
Type: `MemoryMappedGenericWatchdog`.

**FVP_BaseR_Cortex_R82R1.bp.hdlcd0**

ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).
Type: `PL370_HDLCD`.

**FVP_BaseR_Cortex_R82R1.bp.hdlcd0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R82R1.bp.hdlcd0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R82R1.bp.hdlcd0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R82R1.bp.hdlcd0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R82R1.bp.hdlcd0_labeller**

Type: `Labeller`.

**FVP_BaseR_Cortex_R82R1.bp.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_BaseR_Cortex_R82R1.bp.mmc**

Generic Multimedia Card.
Type: `MMC`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_BaseR_Cortex_R82R1.bp.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R82R1.bp.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R82R1.bp.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R82R1.bp.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R82R1.bp.mpe**

Encrypt memory transactions for each encryption context with an independent key to prevent mismatch access.
Type: `PVMemoryProtectionEngine`.

**FVP_BaseR_Cortex_R82R1.bp.nontrustedrom**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_BaseR_Cortex_R82R1.bp.nontrustedromloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_BaseR_Cortex_R82R1.bp.ns_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R82R1.bp.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_BaseR_Cortex_R82R1.bp.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R82R1.bp.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R82R1.bp.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R82R1.bp.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R82R1.bp.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R82R1.bp.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_BaseR_Cortex_R82R1.bp.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R82R1.bp.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R82R1.bp.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R82R1.bp.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R82R1.bp.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R82R1.bp.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_BaseR_Cortex_R82R1.bp.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R82R1.bp.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R82R1.bp.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_BaseR_Cortex_R82R1.bp.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R82R1.bp.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R82R1.bp.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_BaseR_Cortex_R82R1.bp.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R82R1.bp.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R82R1.bp.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R82R1.bp.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R82R1.bp.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R82R1.bp.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_BaseR_Cortex_R82R1.bp.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R82R1.bp.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R82R1.bp.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R82R1.bp.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R82R1.bp.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_BaseR_Cortex_R82R1.bp.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R82R1.bp.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_BaseR_Cortex_R82R1.bp.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R82R1.bp.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R82R1.bp.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_BaseR_Cortex_R82R1.bp.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R82R1.bp.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_BaseR_Cortex_R82R1.bp.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R82R1.bp.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_BaseR_Cortex_R82R1.bp.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

**FVP_BaseR_Cortex_R82R1.bp.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R82R1.bp.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R82R1.bp.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R82R1.bp.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R82R1.bp.pl111_clcd_labeller**

Type: `Labeller`.

**FVP_BaseR_Cortex_R82R1.bp.pl180_mci**

ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

**FVP_BaseR_Cortex_R82R1.bp.ps2keyboard**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

**FVP_BaseR_Cortex_R82R1.bp.ps2keyboard.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R82R1.bp.ps2keyboard.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R82R1.bp.ps2keyboard.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_BaseR_Cortex_R82R1.bp.ps2keyboard.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R82R1.bp.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

**FVP_BaseR_Cortex_R82R1.bp.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R82R1.bp.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R82R1.bp.ps2mouse.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R82R1.bp.ps2mouse.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R82R1.bp.psram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R82R1.bp.refcounter**

Memory Mapped Counter Module for Generic Timers.
Type: `MemoryMappedCounterModule`.

**FVP_BaseR_Cortex_R82R1.bp.reset_or**

Or Gate.
Type: `OrGate`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_BaseR_Cortex_R82R1.bp.rl_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R82R1.bp.rt_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R82R1.bp.s_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R82R1.bp.secureDRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R82R1.bp.secureSRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R82R1.bp.secureflash**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_BaseR_Cortex_R82R1.bp.secureflashloader**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_BaseR_Cortex_R82R1.bp.smsc_91c111**
SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_BaseR_Cortex_R82R1.bp.sp805_wdog**
ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_BaseR_Cortex_R82R1.bp.sp805_wdog.clocktimer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R82R1.bp.sp805_wdog.clocktimer.timer**
A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R82R1.bp.sp805_wdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R82R1.bp.sp805_wdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R82R1.bp.sp810_sysctrl**

Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_BaseR_Cortex_R82R1.bp.sp810_sysctrl.clkdiv_clk0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R82R1.bp.sp810_sysctrl.clkdiv_clk1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R82R1.bp.sp810_sysctrl.clkdiv_clk2**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R82R1.bp.sp810_sysctrl.clkdiv_clk3**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R82R1.bp.sram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R82R1.bp.terminal_0**

Telnet terminal interface.
Type: `TelnetTerminal`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_BaseR_Cortex_R82R1.bp.terminal_1**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_BaseR_Cortex_R82R1.bp.terminal_2**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_BaseR_Cortex_R82R1.bp.terminal_3**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_BaseR_Cortex_R82R1.bp.trusted_key_storage**

Trusted Root-Key Storage unit.
Type: `RootKeyStorage`.

**FVP_BaseR_Cortex_R82R1.bp.trusted_nv_counter**

Trusted Non-Volatile Counter unit.
Type: `NonVolatileCounter`.

**FVP_BaseR_Cortex_R82R1.bp.trusted_rng**

Random Number Generator unit.
Type: `RandomNumberGenerator`.

**FVP_BaseR_Cortex_R82R1.bp.trusted_watchdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_BaseR_Cortex_R82R1.bp.trusted_watchdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R82R1.bp.trusted_watchdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R82R1.bp.trusted_watchdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_BaseR_Cortex_R82R1.bp.trusted_watchdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R82R1.bp.tzc_400**

TrustZone Address Space Controller.
Type: `TZC_400`.

**FVP_BaseR_Cortex_R82R1.bp.ve_sysregs**

Type: `VE_SysRegs`.

**FVP_BaseR_Cortex_R82R1.bp.vedcc**

Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

**FVP_BaseR_Cortex_R82R1.bp.virtio_net**

VirtioNet device over MMIO transport.
Type: `VirtioNetMMIO`.

**FVP_BaseR_Cortex_R82R1.bp.virtio_net_labeller**

Type: `Labeller`.

**FVP_BaseR_Cortex_R82R1.bp.virtio_rng**

VirtioEntropy device over MMIO transport.
Type: `VirtioEntropyMMIO`.

**FVP_BaseR_Cortex_R82R1.bp.virtioblockdevice**

VirtioBlock device over MMIO transport.
Type: `VirtioBlockMMIO`.

**FVP_BaseR_Cortex_R82R1.bp.virtioblockdevice_labeller**

Type: `Labeller`.

**FVP_BaseR_Cortex_R82R1.bp.virtiop9device**

virtio P9 server.
Type: `VirtioP9Device`.

**FVP_BaseR_Cortex_R82R1.bp.virtiop9device_labeller**

Type: `Labeller`.

**FVP_BaseR_Cortex_R82R1.bp.vis**

Display window for VE using Visualisation library.
Type: `VEVisualisation`.

**FVP_BaseR_Cortex_R82R1.bp.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_BaseR_Cortex_R82R1.bp.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R82R1.bp.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R82R1.bp.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R82R1.bp.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R82R1.bp.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R82R1.bp.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R82R1.bp.vram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R82R1.cci400**

Cache Coherent Interconnect for AXI4 ACE.
Type: `CCI400`.

**FVP_BaseR_Cortex_R82R1.clockdivider0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R82R1.cluster0**
ARM Cortex-R82 Cluster CT model.
Type: `Cluster_ARM_Cortex-R82`.

**FVP_BaseR_Cortex_R82R1.cluster0.cpu0**
ARM Cortex-R82 CT model.
Type: `ARM_Cortex-R82`.

**FVP_BaseR_Cortex_R82R1.cluster0.cpu0.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_BaseR_Cortex_R82R1.cluster0.cpu0.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_BaseR_Cortex_R82R1.cluster0.cpu0.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_BaseR_Cortex_R82R1.cluster0.cpu1**
ARM Cortex-R82 CT model.
Type: `ARM_Cortex-R82`.

**FVP_BaseR_Cortex_R82R1.cluster0.cpu1.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_BaseR_Cortex_R82R1.cluster0.cpu1.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_BaseR_Cortex_R82R1.cluster0.cpu1.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_BaseR_Cortex_R82R1.cluster0.cpu2**
ARM Cortex-R82 CT model.
Type: `ARM_Cortex-R82`.

**FVP_BaseR_Cortex_R82R1.cluster0.cpu2.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_BaseR_Cortex_R82R1.cluster0.cpu2.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_BaseR_Cortex_R82R1.cluster0.cpu2.l1icache**
PV Cache.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `PVCache`.

**FVP_BaseR_Cortex_R82R1.cluster0.cpu3**

ARM Cortex-R82 CT model.
Type: `ARM_Cortex-R82`.

**FVP_BaseR_Cortex_R82R1.cluster0.cpu3.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_BaseR_Cortex_R82R1.cluster0.cpu3.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_BaseR_Cortex_R82R1.cluster0.cpu3.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_BaseR_Cortex_R82R1.cluster0.cpu4**

ARM Cortex-R82 CT model.
Type: `ARM_Cortex-R82`.

**FVP_BaseR_Cortex_R82R1.cluster0.cpu4.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_BaseR_Cortex_R82R1.cluster0.cpu4.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_BaseR_Cortex_R82R1.cluster0.cpu4.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_BaseR_Cortex_R82R1.cluster0.cpu5**

ARM Cortex-R82 CT model.
Type: `ARM_Cortex-R82`.

**FVP_BaseR_Cortex_R82R1.cluster0.cpu5.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_BaseR_Cortex_R82R1.cluster0.cpu5.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_BaseR_Cortex_R82R1.cluster0.cpu5.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_BaseR_Cortex_R82R1.cluster0.cpu6**

ARM Cortex-R82 CT model.
Type: `ARM_Cortex-R82`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_BaseR_Cortex_R82R1.cluster0.cpu6.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_BaseR_Cortex_R82R1.cluster0.cpu6.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_BaseR_Cortex_R82R1.cluster0.cpu6.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_BaseR_Cortex_R82R1.cluster0.cpu7**
ARM Cortex-R82 CT model.
Type: `ARM_Cortex-R82`.

**FVP_BaseR_Cortex_R82R1.cluster0.cpu7.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_BaseR_Cortex_R82R1.cluster0.cpu7.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_BaseR_Cortex_R82R1.cluster0.cpu7.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_BaseR_Cortex_R82R1.cluster0.l2_cache**
PV Cache.
Type: `PVCache`.

**FVP_BaseR_Cortex_R82R1.cluster0_labeller**
Type: `Labeller`.

**FVP_BaseR_Cortex_R82R1.dapmemlogger**
Bus Logger.
Type: `PVBusLogger`.

**FVP_BaseR_Cortex_R82R1.elfloader**
ELF loader component.
Type: `ElfLoader`.

**FVP_BaseR_Cortex_R82R1.gic_distributor**
GIC Interrupt Redistribution Infrastucture component.
Type: `GIC_IRI`.

**FVP_BaseR_Cortex_R82R1.llram0**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R82R1.pctl**
Base Platforms Power Controller.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `Base_PowerController`.

**FVP_BaseR_Cortex_R82R1.pctl.timer_reset**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R82R1.pctl.timer_reset.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R82R1.pctl.timer_reset.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R82R1.pctl.timer_reset.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

## 14.6 FVP_BaseR_Cortex-R82x1

List of instances in FVP_BaseR_Cortex-R82x1.

### FVP_BaseR_Cortex-R82x1 instances

**FVP_BaseR_Cortex_R82x1**

Base Platform Compute Subsystem for ARMCortexR82x1CT.
Type: `FVP_BaseR_Cortex_R82x1`.

**FVP_BaseR_Cortex_R82x1.bp**

Peripherals and address map for the Base Platform.
Type: `BasePlatformPeripherals`.

**FVP_BaseR_Cortex_R82x1.bp.Timer_0_1**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_BaseR_Cortex_R82x1.bp.Timer_0_1.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R82x1.bp.Timer_0_1.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R82x1.bp.Timer_0_1.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_BaseR_Cortex_R82x1.bp.Timer_0_1.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_BaseR_Cortex_R82x1.bp.Timer_2_3**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_BaseR_Cortex_R82x1.bp.Timer_2_3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R82x1.bp.Timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R82x1.bp.Timer_2_3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_BaseR_Cortex_R82x1.bp.Timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_BaseR_Cortex_R82x1.bp.ap_refclk**

ARM Generic Timer.
Type: `MemoryMappedGenericTimer`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_BaseR_Cortex_R82x1.bp.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**FVP_BaseR_Cortex_R82x1.bp.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R82x1.bp.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R82x1.bp.clock300MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R82x1.bp.clock32KHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R82x1.bp.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R82x1.bp.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R82x1.bp.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ClockDivider`.

**FVP_BaseR_Cortex_R82x1.bp.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R82x1.bp.dmc**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R82x1.bp.dmc_phy**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R82x1.bp.dummy_local_dap_rom**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R82x1.bp.dummy_ram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R82x1.bp.dummy_usb**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R82x1.bp.exclusive_monitor**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_BaseR_Cortex_R82x1.bp.flash0**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_BaseR_Cortex_R82x1.bp.flash1**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_BaseR_Cortex_R82x1.bp.flashloader0**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_BaseR_Cortex_R82x1.bp.flashloader1**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_BaseR_Cortex_R82x1.bp.generic_watchdog**

ARM Generic Watchdog.
Type: `MemoryMappedGenericWatchdog`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_BaseR_Cortex_R82x1.bp.hdlcd0**

ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).
Type: `PL370_HDLCD`.

**FVP_BaseR_Cortex_R82x1.bp.hdlcd0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R82x1.bp.hdlcd0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R82x1.bp.hdlcd0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R82x1.bp.hdlcd0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R82x1.bp.hdlcd0_labeller**

Type: `Labeller`.

**FVP_BaseR_Cortex_R82x1.bp.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_BaseR_Cortex_R82x1.bp.mmc**

Generic Multimedia Card.
Type: `MMC`.

**FVP_BaseR_Cortex_R82x1.bp.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R82x1.bp.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R82x1.bp.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R82x1.bp.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R82x1.bp.mpe**

Encrypt memory transactions for each encryption context with an independent key to prevent mismatch access.
Type: `PVMemoryProtectionEngine`.

**FVP_BaseR_Cortex_R82x1.bp.nontrustedrom**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_BaseR_Cortex_R82x1.bp.nontrustedromloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_BaseR_Cortex_R82x1.bp.ns_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R82x1.bp.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_BaseR_Cortex_R82x1.bp.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R82x1.bp.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R82x1.bp.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R82x1.bp.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R82x1.bp.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R82x1.bp.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_BaseR_Cortex_R82x1.bp.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R82x1.bp.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R82x1.bp.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R82x1.bp.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R82x1.bp.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R82x1.bp.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_BaseR_Cortex_R82x1.bp.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R82x1.bp.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R82x1.bp.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R82x1.bp.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R82x1.bp.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_BaseR_Cortex_R82x1.bp.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_BaseR_Cortex_R82x1.bp.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R82x1.bp.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R82x1.bp.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R82x1.bp.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R82x1.bp.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R82x1.bp.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

**FVP_BaseR_Cortex_R82x1.bp.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R82x1.bp.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R82x1.bp.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R82x1.bp.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R82x1.bp.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_BaseR_Cortex_R82x1.bp.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R82x1.bp.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R82x1.bp.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R82x1.bp.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_BaseR_Cortex_R82x1.bp.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_BaseR_Cortex_R82x1.bp.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R82x1.bp.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_BaseR_Cortex_R82x1.bp.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R82x1.bp.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_BaseR_Cortex_R82x1.bp.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

**FVP_BaseR_Cortex_R82x1.bp.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R82x1.bp.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R82x1.bp.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R82x1.bp.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R82x1.bp.pl111_clcd_labeller**

Type: `Labeller`.

**FVP_BaseR_Cortex_R82x1.bp.pl180_mci**

ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

**FVP_BaseR_Cortex_R82x1.bp.ps2keyboard**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

**FVP_BaseR_Cortex_R82x1.bp.ps2keyboard.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R82x1.bp.ps2keyboard.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R82x1.bp.ps2keyboard.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R82x1.bp.ps2keyboard.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R82x1.bp.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `PS2Mouse`.

**FVP_BaseR_Cortex_R82x1.bp.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R82x1.bp.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R82x1.bp.ps2mouse.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R82x1.bp.ps2mouse.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R82x1.bp.psram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R82x1.bp.refcounter**

Memory Mapped Counter Module for Generic Timers.
Type: `MemoryMappedCounterModule`.

**FVP_BaseR_Cortex_R82x1.bp.reset_or**

Or Gate.
Type: `OrGate`.

**FVP_BaseR_Cortex_R82x1.bp.rl_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R82x1.bp.rt_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R82x1.bp.s_dram**

RAM device, can be dynamic or static ram.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `RAMDevice`.

**FVP_BaseR_Cortex_R82x1.bp.secureDRAM**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R82x1.bp.secureSRAM**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R82x1.bp.secureflash**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_BaseR_Cortex_R82x1.bp.secureflashloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_BaseR_Cortex_R82x1.bp.smsc_91c111**

SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_BaseR_Cortex_R82x1.bp.sp805_wdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_BaseR_Cortex_R82x1.bp.sp805_wdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R82x1.bp.sp805_wdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R82x1.bp.sp805_wdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R82x1.bp.sp805_wdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R82x1.bp.sp810_sysctrl**

Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_BaseR_Cortex_R82x1.bp.sp810_sysctrl.clkdiv_clk0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R82x1.bp.sp810_sysctrl.clkdiv_clk1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R82x1.bp.sp810_sysctrl.clkdiv_clk2**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R82x1.bp.sp810_sysctrl.clkdiv_clk3**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R82x1.bp.sram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R82x1.bp.terminal_0**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_BaseR_Cortex_R82x1.bp.terminal_1**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_BaseR_Cortex_R82x1.bp.terminal_2**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_BaseR_Cortex_R82x1.bp.terminal_3**

Telnet terminal interface.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `TelnetTerminal`.

**FVP_BaseR_Cortex_R82x1.bp.trusted_key_storage**

Trusted Root-Key Storage unit.
Type: `RootKeyStorage`.

**FVP_BaseR_Cortex_R82x1.bp.trusted_nv_counter**

Trusted Non-Volatile Counter unit.
Type: `NonVolatileCounter`.

**FVP_BaseR_Cortex_R82x1.bp.trusted_rng**

Random Number Generator unit.
Type: `RandomNumberGenerator`.

**FVP_BaseR_Cortex_R82x1.bp.trusted_watchdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_BaseR_Cortex_R82x1.bp.trusted_watchdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R82x1.bp.trusted_watchdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R82x1.bp.trusted_watchdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R82x1.bp.trusted_watchdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R82x1.bp.tzc_400**

TrustZone Address Space Controller.
Type: `TZC_400`.

**FVP_BaseR_Cortex_R82x1.bp.ve_sysregs**

Type: `VE_SysRegs`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_BaseR_Cortex_R82x1.bp.vedcc**

Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

**FVP_BaseR_Cortex_R82x1.bp.virtio_net**

VirtioNet device over MMIO transport.
Type: `VirtioNetMMIO`.

**FVP_BaseR_Cortex_R82x1.bp.virtio_net_labeller**

Type: `Labeller`.

**FVP_BaseR_Cortex_R82x1.bp.virtio_rng**

VirtioEntropy device over MMIO transport.
Type: `VirtioEntropyMMIO`.

**FVP_BaseR_Cortex_R82x1.bp.virtioblockdevice**

VirtioBlock device over MMIO transport.
Type: `VirtioBlockMMIO`.

**FVP_BaseR_Cortex_R82x1.bp.virtioblockdevice_labeller**

Type: `Labeller`.

**FVP_BaseR_Cortex_R82x1.bp.virtiop9device**

virtio P9 server.
Type: `VirtioP9Device`.

**FVP_BaseR_Cortex_R82x1.bp.virtiop9device_labeller**

Type: `Labeller`.

**FVP_BaseR_Cortex_R82x1.bp.vis**

Display window for VE using Visualisation library.
Type: `VEVisualisation`.

**FVP_BaseR_Cortex_R82x1.bp.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_BaseR_Cortex_R82x1.bp.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R82x1.bp.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R82x1.bp.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R82x1.bp.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R82x1.bp.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R82x1.bp.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R82x1.bp.vram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R82x1.cci400**

Cache Coherent Interconnect for AXI4 ACE.
Type: `CCI400`.

**FVP_BaseR_Cortex_R82x1.clockdivider0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R82x1.cluster0**

ARM Cortex-R82 Cluster CT model.
Type: `Cluster_ARM_Cortex-R82`.

**FVP_BaseR_Cortex_R82x1.cluster0.cpu0**

ARM Cortex-R82 CT model.
Type: `ARM_Cortex-R82`.

**FVP_BaseR_Cortex_R82x1.cluster0.cpu0.dtlb**

TLB - instruction, data or unified.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `TlbCadi`.

**FVP_BaseR_Cortex_R82x1.cluster0.cpu0.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_BaseR_Cortex_R82x1.cluster0.cpu0.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_BaseR_Cortex_R82x1.cluster0.l2_cache**

PV Cache.
Type: `PVCache`.

**FVP_BaseR_Cortex_R82x1.cluster0_labeller**

Type: `Labeller`.

**FVP_BaseR_Cortex_R82x1.dapmemlogger**

Bus Logger.
Type: `PVBusLogger`.

**FVP_BaseR_Cortex_R82x1.elfloader**

ELF loader component.
Type: `ElfLoader`.

**FVP_BaseR_Cortex_R82x1.gic_distributor**

GIC Interrupt Redistribution Infrastucture component.
Type: `GIC_IRI`.

**FVP_BaseR_Cortex_R82x1.llram0**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R82x1.pctl**

Base Platforms Power Controller.
Type: `Base_PowerController`.

**FVP_BaseR_Cortex_R82x1.pctl.timer_reset**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R82x1.pctl.timer_reset.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**`FVP_BaseR_Cortex_R82x1.pctl.timer_reset.timer.thread`**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**`FVP_BaseR_Cortex_R82x1.pctl.timer_reset.timer.thread_event`**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

## 14.7 FVP_BaseR_Cortex-R82x2

List of instances in FVP_BaseR_Cortex-R82x2.

### FVP_BaseR_Cortex-R82x2 instances

**`FVP_BaseR_Cortex_R82x2`**
Base Platform Compute Subsystem for ARMCortexR82x2CT.
Type: `FVP_BaseR_Cortex_R82x2`.

**`FVP_BaseR_Cortex_R82x2.bp`**
Peripherals and address map for the Base Platform.
Type: `BasePlatformPeripherals`.

**`FVP_BaseR_Cortex_R82x2.bp.Timer_0_1`**
ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**`FVP_BaseR_Cortex_R82x2.bp.Timer_0_1.clk_div0`**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**`FVP_BaseR_Cortex_R82x2.bp.Timer_0_1.clk_div1`**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**`FVP_BaseR_Cortex_R82x2.bp.Timer_0_1.counter0`**
Internal component used by SP804 Timer module.
Type: `CounterModule`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_BaseR_Cortex_R82x2.bp.Timer_0_1.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_BaseR_Cortex_R82x2.bp.Timer_2_3**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_BaseR_Cortex_R82x2.bp.Timer_2_3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R82x2.bp.Timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R82x2.bp.Timer_2_3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_BaseR_Cortex_R82x2.bp.Timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_BaseR_Cortex_R82x2.bp.ap_refclk**

ARM Generic Timer.
Type: `MemoryMappedGenericTimer`.

**FVP_BaseR_Cortex_R82x2.bp.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**FVP_BaseR_Cortex_R82x2.bp.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R82x2.bp.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_BaseR_Cortex_R82x2.bp.clock300MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R82x2.bp.clock32KHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R82x2.bp.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R82x2.bp.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R82x2.bp.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R82x2.bp.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R82x2.bp.dmc**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R82x2.bp.dmc_phy**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_BaseR_Cortex_R82x2.bp.dummy_local_dap_rom**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R82x2.bp.dummy_ram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R82x2.bp.dummy_usb**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R82x2.bp.exclusive_monitor**
Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_BaseR_Cortex_R82x2.bp.flash0**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_BaseR_Cortex_R82x2.bp.flash1**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_BaseR_Cortex_R82x2.bp.flashloader0**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_BaseR_Cortex_R82x2.bp.flashloader1**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_BaseR_Cortex_R82x2.bp.generic_watchdog**
ARM Generic Watchdog.
Type: `MemoryMappedGenericWatchdog`.

**FVP_BaseR_Cortex_R82x2.bp.hdlcd0**
ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).
Type: `PL370_HDLCD`.

**FVP_BaseR_Cortex_R82x2.bp.hdlcd0.timer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R82x2.bp.hdlcd0.timer.timer**
A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R82x2.bp.hdlcd0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R82x2.bp.hdlcd0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R82x2.bp.hdlcd0_labeller**

Type: `Labeller`.

**FVP_BaseR_Cortex_R82x2.bp.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_BaseR_Cortex_R82x2.bp.mmc**

Generic Multimedia Card.
Type: `MMC`.

**FVP_BaseR_Cortex_R82x2.bp.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R82x2.bp.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R82x2.bp.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R82x2.bp.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R82x2.bp.mpe**

Encrypt memory transactions for each encryption context with an independent key to prevent mismatch access.
Type: `PVMemoryProtectionEngine`.

**FVP_BaseR_Cortex_R82x2.bp.nontrustedrom**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_BaseR_Cortex_R82x2.bp.nontrustedromloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_BaseR_Cortex_R82x2.bp.ns_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R82x2.bp.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_BaseR_Cortex_R82x2.bp.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R82x2.bp.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R82x2.bp.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R82x2.bp.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_BaseR_Cortex_R82x2.bp.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R82x2.bp.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_BaseR_Cortex_R82x2.bp.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R82x2.bp.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R82x2.bp.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R82x2.bp.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R82x2.bp.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R82x2.bp.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_BaseR_Cortex_R82x2.bp.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R82x2.bp.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R82x2.bp.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R82x2.bp.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R82x2.bp.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R82x2.bp.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_BaseR_Cortex_R82x2.bp.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R82x2.bp.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R82x2.bp.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R82x2.bp.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R82x2.bp.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R82x2.bp.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

**FVP_BaseR_Cortex_R82x2.bp.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R82x2.bp.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R82x2.bp.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R82x2.bp.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_BaseR_Cortex_R82x2.bp.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_BaseR_Cortex_R82x2.bp.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R82x2.bp.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R82x2.bp.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R82x2.bp.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R82x2.bp.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_BaseR_Cortex_R82x2.bp.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R82x2.bp.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_BaseR_Cortex_R82x2.bp.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R82x2.bp.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_BaseR_Cortex_R82x2.bp.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

**FVP_BaseR_Cortex_R82x2.bp.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R82x2.bp.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R82x2.bp.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R82x2.bp.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R82x2.bp.pl111_clcd_labeller**

Type: `Labeller`.

**FVP_BaseR_Cortex_R82x2.bp.pl180_mci**

ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

**FVP_BaseR_Cortex_R82x2.bp.ps2keyboard**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_BaseR_Cortex_R82x2.bp.ps2keyboard.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R82x2.bp.ps2keyboard.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R82x2.bp.ps2keyboard.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R82x2.bp.ps2keyboard.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R82x2.bp.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

**FVP_BaseR_Cortex_R82x2.bp.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R82x2.bp.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R82x2.bp.ps2mouse.ps2_clocktimer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R82x2.bp.ps2mouse.ps2_clocktimer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R82x2.bp.psram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R82x2.bp.refcounter**
Memory Mapped Counter Module for Generic Timers.
Type: `MemoryMappedCounterModule`.

**FVP_BaseR_Cortex_R82x2.bp.reset_or**
Or Gate.
Type: `OrGate`.

**FVP_BaseR_Cortex_R82x2.bp.rl_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R82x2.bp.rt_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R82x2.bp.s_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R82x2.bp.secureDRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R82x2.bp.secureSRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R82x2.bp.secureflash**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_BaseR_Cortex_R82x2.bp.secureflashloader**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_BaseR_Cortex_R82x2.bp.smsc_91c111**

SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_BaseR_Cortex_R82x2.bp.sp805_wdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_BaseR_Cortex_R82x2.bp.sp805_wdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R82x2.bp.sp805_wdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R82x2.bp.sp805_wdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R82x2.bp.sp805_wdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R82x2.bp.sp810_sysctrl**

Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_BaseR_Cortex_R82x2.bp.sp810_sysctrl.clkdiv_clk0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R82x2.bp.sp810_sysctrl.clkdiv_clk1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R82x2.bp.sp810_sysctrl.clkdiv_clk2**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R82x2.bp.sp810_sysctrl.clkdiv_clk3**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R82x2.bp.sram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R82x2.bp.terminal_0**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_BaseR_Cortex_R82x2.bp.terminal_1**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_BaseR_Cortex_R82x2.bp.terminal_2**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_BaseR_Cortex_R82x2.bp.terminal_3**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_BaseR_Cortex_R82x2.bp.trusted_key_storage**

Trusted Root-Key Storage unit.
Type: `RootKeyStorage`.

**FVP_BaseR_Cortex_R82x2.bp.trusted_nv_counter**

Trusted Non-Volatile Counter unit.
Type: `NonVolatileCounter`.

**FVP_BaseR_Cortex_R82x2.bp.trusted_rng**

Random Number Generator unit.
Type: `RandomNumberGenerator`.

**FVP_BaseR_Cortex_R82x2.bp.trusted_watchdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_BaseR_Cortex_R82x2.bp.trusted_watchdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R82x2.bp.trusted_watchdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R82x2.bp.trusted_watchdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R82x2.bp.trusted_watchdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R82x2.bp.tzc_400**

TrustZone Address Space Controller.
Type: `TZC_400`.

**FVP_BaseR_Cortex_R82x2.bp.ve_sysregs**

Type: `VE_SysRegs`.

**FVP_BaseR_Cortex_R82x2.bp.vedcc**

Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

**FVP_BaseR_Cortex_R82x2.bp.virtio_net**

VirtioNet device over MMIO transport.
Type: `VirtioNetMMIO`.

**FVP_BaseR_Cortex_R82x2.bp.virtio_net_labeller**

Type: `Labeller`.

**FVP_BaseR_Cortex_R82x2.bp.virtio_rng**

VirtioEntropy device over MMIO transport.
Type: `VirtioEntropyMMIO`.

**FVP_BaseR_Cortex_R82x2.bp.virtioblockdevice**

VirtioBlock device over MMIO transport.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `VirtioBlockMMIO`.

**FVP_BaseR_Cortex_R82x2.bp.virtioblockdevice_labeller**

Type: `Labeller`.

**FVP_BaseR_Cortex_R82x2.bp.virtiop9device**

virtio P9 server.
Type: `VirtioP9Device`.

**FVP_BaseR_Cortex_R82x2.bp.virtiop9device_labeller**

Type: `Labeller`.

**FVP_BaseR_Cortex_R82x2.bp.vis**

Display window for VE using Visualisation library.
Type: `VEVisualisation`.

**FVP_BaseR_Cortex_R82x2.bp.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_BaseR_Cortex_R82x2.bp.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R82x2.bp.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R82x2.bp.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R82x2.bp.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_BaseR_Cortex_R82x2.bp.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R82x2.bp.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R82x2.bp.vram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R82x2.cci400**

Cache Coherent Interconnect for AXI4 ACE.
Type: `CCI400`.

**FVP_BaseR_Cortex_R82x2.clockdivider0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R82x2.cluster0**

ARM Cortex-R82 Cluster CT model.
Type: `Cluster_ARM_Cortex-R82`.

**FVP_BaseR_Cortex_R82x2.cluster0.cpu0**

ARM Cortex-R82 CT model.
Type: `ARM_Cortex-R82`.

**FVP_BaseR_Cortex_R82x2.cluster0.cpu0.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_BaseR_Cortex_R82x2.cluster0.cpu0.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_BaseR_Cortex_R82x2.cluster0.cpu0.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_BaseR_Cortex_R82x2.cluster0.cpu1**

ARM Cortex-R82 CT model.
Type: `ARM_Cortex-R82`.

**FVP_BaseR_Cortex_R82x2.cluster0.cpu1.dtlb**

TLB - instruction, data or unified.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `TlbCadi`.

**FVP_BaseR_Cortex_R82x2.cluster0.cpu1.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_BaseR_Cortex_R82x2.cluster0.cpu1.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_BaseR_Cortex_R82x2.cluster0.l2_cache**

PV Cache.
Type: `PVCache`.

**FVP_BaseR_Cortex_R82x2.cluster0_labeller**

Type: `Labeller`.

**FVP_BaseR_Cortex_R82x2.dapmemlogger**

Bus Logger.
Type: `PVBusLogger`.

**FVP_BaseR_Cortex_R82x2.elfloader**

ELF loader component.
Type: `ElfLoader`.

**FVP_BaseR_Cortex_R82x2.gic_distributor**

GIC Interrupt Redistribution Infrastucture component.
Type: `GIC_IRI`.

**FVP_BaseR_Cortex_R82x2.llram0**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R82x2.pctl**

Base Platforms Power Controller.
Type: `Base_PowerController`.

**FVP_BaseR_Cortex_R82x2.pctl.timer_reset**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R82x2.pctl.timer_reset.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**`FVP_BaseR_Cortex_R82x2.pctl.timer_reset.timer.thread`**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**`FVP_BaseR_Cortex_R82x2.pctl.timer_reset.timer.thread_event`**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

## 14.8 FVP_BaseR_Cortex-R82x4

List of instances in FVP_BaseR_Cortex-R82x4.

### FVP_BaseR_Cortex-R82x4 instances

**`FVP_BaseR_Cortex_R82x4`**
Base Platform Compute Subsystem for ARMCortexR82x4CT.
Type: `FVP_BaseR_Cortex_R82x4`.

**`FVP_BaseR_Cortex_R82x4.bp`**
Peripherals and address map for the Base Platform.
Type: `BasePlatformPeripherals`.

**`FVP_BaseR_Cortex_R82x4.bp.Timer_0_1`**
ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**`FVP_BaseR_Cortex_R82x4.bp.Timer_0_1.clk_div0`**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**`FVP_BaseR_Cortex_R82x4.bp.Timer_0_1.clk_div1`**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**`FVP_BaseR_Cortex_R82x4.bp.Timer_0_1.counter0`**
Internal component used by SP804 Timer module.
Type: `CounterModule`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_BaseR_Cortex_R82x4.bp.Timer_0_1.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_BaseR_Cortex_R82x4.bp.Timer_2_3**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_BaseR_Cortex_R82x4.bp.Timer_2_3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R82x4.bp.Timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R82x4.bp.Timer_2_3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_BaseR_Cortex_R82x4.bp.Timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_BaseR_Cortex_R82x4.bp.ap_refclk**

ARM Generic Timer.
Type: `MemoryMappedGenericTimer`.

**FVP_BaseR_Cortex_R82x4.bp.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**FVP_BaseR_Cortex_R82x4.bp.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R82x4.bp.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_BaseR_Cortex_R82x4.bp.clock300MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R82x4.bp.clock32KHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R82x4.bp.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R82x4.bp.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R82x4.bp.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R82x4.bp.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R82x4.bp.dmc**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R82x4.bp.dmc_phy**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_BaseR_Cortex_R82x4.bp.dummy_local_dap_rom**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R82x4.bp.dummy_ram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R82x4.bp.dummy_usb**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R82x4.bp.exclusive_monitor**
Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_BaseR_Cortex_R82x4.bp.flash0**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_BaseR_Cortex_R82x4.bp.flash1**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_BaseR_Cortex_R82x4.bp.flashloader0**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_BaseR_Cortex_R82x4.bp.flashloader1**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_BaseR_Cortex_R82x4.bp.generic_watchdog**
ARM Generic Watchdog.
Type: `MemoryMappedGenericWatchdog`.

**FVP_BaseR_Cortex_R82x4.bp.hdlcd0**
ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).
Type: `PL370_HDLCD`.

**FVP_BaseR_Cortex_R82x4.bp.hdlcd0.timer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R82x4.bp.hdlcd0.timer.timer**
A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R82x4.bp.hdlcd0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R82x4.bp.hdlcd0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R82x4.bp.hdlcd0_labeller**

Type: `Labeller`.

**FVP_BaseR_Cortex_R82x4.bp.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_BaseR_Cortex_R82x4.bp.mmc**

Generic Multimedia Card.
Type: `MMC`.

**FVP_BaseR_Cortex_R82x4.bp.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R82x4.bp.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R82x4.bp.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R82x4.bp.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R82x4.bp.mpe**

Encrypt memory transactions for each encryption context with an independent key to prevent mismatch access.
Type: `PVMemoryProtectionEngine`.

**FVP_BaseR_Cortex_R82x4.bp.nontrustedrom**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_BaseR_Cortex_R82x4.bp.nontrustedromloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_BaseR_Cortex_R82x4.bp.ns_dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R82x4.bp.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_BaseR_Cortex_R82x4.bp.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R82x4.bp.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R82x4.bp.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R82x4.bp.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_BaseR_Cortex_R82x4.bp.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R82x4.bp.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_BaseR_Cortex_R82x4.bp.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R82x4.bp.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R82x4.bp.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R82x4.bp.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R82x4.bp.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R82x4.bp.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_BaseR_Cortex_R82x4.bp.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R82x4.bp.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R82x4.bp.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R82x4.bp.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R82x4.bp.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R82x4.bp.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_BaseR_Cortex_R82x4.bp.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R82x4.bp.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R82x4.bp.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R82x4.bp.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R82x4.bp.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R82x4.bp.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

**FVP_BaseR_Cortex_R82x4.bp.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R82x4.bp.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R82x4.bp.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R82x4.bp.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_BaseR_Cortex_R82x4.bp.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_BaseR_Cortex_R82x4.bp.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R82x4.bp.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R82x4.bp.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R82x4.bp.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R82x4.bp.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_BaseR_Cortex_R82x4.bp.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R82x4.bp.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_BaseR_Cortex_R82x4.bp.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R82x4.bp.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_BaseR_Cortex_R82x4.bp.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

**FVP_BaseR_Cortex_R82x4.bp.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R82x4.bp.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R82x4.bp.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R82x4.bp.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R82x4.bp.pl111_clcd_labeller**

Type: `Labeller`.

**FVP_BaseR_Cortex_R82x4.bp.pl180_mci**

ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

**FVP_BaseR_Cortex_R82x4.bp.ps2keyboard**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_BaseR_Cortex_R82x4.bp.ps2keyboard.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R82x4.bp.ps2keyboard.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R82x4.bp.ps2keyboard.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R82x4.bp.ps2keyboard.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R82x4.bp.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

**FVP_BaseR_Cortex_R82x4.bp.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R82x4.bp.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R82x4.bp.ps2mouse.ps2_clocktimer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R82x4.bp.ps2mouse.ps2_clocktimer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R82x4.bp.psram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R82x4.bp.refcounter**
Memory Mapped Counter Module for Generic Timers.
Type: `MemoryMappedCounterModule`.

**FVP_BaseR_Cortex_R82x4.bp.reset_or**
Or Gate.
Type: `OrGate`.

**FVP_BaseR_Cortex_R82x4.bp.rl_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R82x4.bp.rt_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R82x4.bp.s_dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R82x4.bp.secureDRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R82x4.bp.secureSRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R82x4.bp.secureflash**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_BaseR_Cortex_R82x4.bp.secureflashloader**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_BaseR_Cortex_R82x4.bp.smsc_91c111**

SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_BaseR_Cortex_R82x4.bp.sp805_wdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_BaseR_Cortex_R82x4.bp.sp805_wdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R82x4.bp.sp805_wdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R82x4.bp.sp805_wdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R82x4.bp.sp805_wdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R82x4.bp.sp810_sysctrl**

Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_BaseR_Cortex_R82x4.bp.sp810_sysctrl.clkdiv_clk0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R82x4.bp.sp810_sysctrl.clkdiv_clk1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R82x4.bp.sp810_sysctrl.clkdiv_clk2**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R82x4.bp.sp810_sysctrl.clkdiv_clk3**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R82x4.bp.sram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R82x4.bp.terminal_0**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_BaseR_Cortex_R82x4.bp.terminal_1**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_BaseR_Cortex_R82x4.bp.terminal_2**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_BaseR_Cortex_R82x4.bp.terminal_3**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_BaseR_Cortex_R82x4.bp.trusted_key_storage**

Trusted Root-Key Storage unit.
Type: `RootKeyStorage`.

**FVP_BaseR_Cortex_R82x4.bp.trusted_nv_counter**

Trusted Non-Volatile Counter unit.
Type: `NonVolatileCounter`.

**FVP_BaseR_Cortex_R82x4.bp.trusted_rng**

Random Number Generator unit.
Type: `RandomNumberGenerator`.

**FVP_BaseR_Cortex_R82x4.bp.trusted_watchdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_BaseR_Cortex_R82x4.bp.trusted_watchdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R82x4.bp.trusted_watchdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R82x4.bp.trusted_watchdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R82x4.bp.trusted_watchdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R82x4.bp.tzc_400**

TrustZone Address Space Controller.
Type: `TZC_400`.

**FVP_BaseR_Cortex_R82x4.bp.ve_sysregs**

Type: `VE_SysRegs`.

**FVP_BaseR_Cortex_R82x4.bp.vedcc**

Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

**FVP_BaseR_Cortex_R82x4.bp.virtio_net**

VirtioNet device over MMIO transport.
Type: `VirtioNetMMIO`.

**FVP_BaseR_Cortex_R82x4.bp.virtio_net_labeller**

Type: `Labeller`.

**FVP_BaseR_Cortex_R82x4.bp.virtio_rng**

VirtioEntropy device over MMIO transport.
Type: `VirtioEntropyMMIO`.

**FVP_BaseR_Cortex_R82x4.bp.virtioblockdevice**

VirtioBlock device over MMIO transport.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `VirtioBlockMMIO`.

**FVP_BaseR_Cortex_R82x4.bp.virtioblockdevice_labeller**

Type: `Labeller`.

**FVP_BaseR_Cortex_R82x4.bp.virtiop9device**

virtio P9 server.
Type: `VirtioP9Device`.

**FVP_BaseR_Cortex_R82x4.bp.virtiop9device_labeller**

Type: `Labeller`.

**FVP_BaseR_Cortex_R82x4.bp.vis**

Display window for VE using Visualisation library.
Type: `VEVisualisation`.

**FVP_BaseR_Cortex_R82x4.bp.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_BaseR_Cortex_R82x4.bp.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R82x4.bp.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R82x4.bp.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R82x4.bp.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_BaseR_Cortex_R82x4.bp.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_BaseR_Cortex_R82x4.bp.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R82x4.bp.vram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R82x4.cci400**

Cache Coherent Interconnect for AXI4 ACE.
Type: `CCI400`.

**FVP_BaseR_Cortex_R82x4.clockdivider0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_BaseR_Cortex_R82x4.cluster0**

ARM Cortex-R82 Cluster CT model.
Type: `Cluster_ARM_Cortex-R82`.

**FVP_BaseR_Cortex_R82x4.cluster0.cpu0**

ARM Cortex-R82 CT model.
Type: `ARM_Cortex-R82`.

**FVP_BaseR_Cortex_R82x4.cluster0.cpu0.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_BaseR_Cortex_R82x4.cluster0.cpu0.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_BaseR_Cortex_R82x4.cluster0.cpu0.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_BaseR_Cortex_R82x4.cluster0.cpu1**

ARM Cortex-R82 CT model.
Type: `ARM_Cortex-R82`.

**FVP_BaseR_Cortex_R82x4.cluster0.cpu1.dtlb**

TLB - instruction, data or unified.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `TlbCadi`.

**FVP_BaseR_Cortex_R82x4.cluster0.cpu1.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_BaseR_Cortex_R82x4.cluster0.cpu1.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_BaseR_Cortex_R82x4.cluster0.cpu2**

ARM Cortex-R82 CT model.
Type: `ARM_Cortex-R82`.

**FVP_BaseR_Cortex_R82x4.cluster0.cpu2.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_BaseR_Cortex_R82x4.cluster0.cpu2.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_BaseR_Cortex_R82x4.cluster0.cpu2.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_BaseR_Cortex_R82x4.cluster0.cpu3**

ARM Cortex-R82 CT model.
Type: `ARM_Cortex-R82`.

**FVP_BaseR_Cortex_R82x4.cluster0.cpu3.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_BaseR_Cortex_R82x4.cluster0.cpu3.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_BaseR_Cortex_R82x4.cluster0.cpu3.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_BaseR_Cortex_R82x4.cluster0.l2_cache**

PV Cache.
Type: `PVCache`.

**FVP_BaseR_Cortex_R82x4.cluster0_labeller**

Type: `Labeller`.

**FVP_BaseR_Cortex_R82x4.dapmemlogger**

Bus Logger.
Type: `PVBusLogger`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_BaseR_Cortex_R82x4.elfloader**

ELF loader component.
Type: `ElfLoader`.

**FVP_BaseR_Cortex_R82x4.gic_distributor**

GIC Interrupt Redistribution Infrastucture component.
Type: `GIC_IRI`.

**FVP_BaseR_Cortex_R82x4.llram0**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_BaseR_Cortex_R82x4.pctl**

Base Platforms Power Controller.
Type: `Base_PowerController`.

**FVP_BaseR_Cortex_R82x4.pctl.timer_reset**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_BaseR_Cortex_R82x4.pctl.timer_reset.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_BaseR_Cortex_R82x4.pctl.timer_reset.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_BaseR_Cortex_R82x4.pctl.timer_reset.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

# 15. VE Platform FVPs

This chapter describes the VE Platform FVPs contained in the FVP Standard Library.

For the VE memory maps, see VE memory map for Cortex-A series and VE memory map for Cortex-R series in the Fast Models Reference Guide.

## 15.1 FVP_VE_Cortex-A15x1

List of instances in FVP_VE_Cortex-A15x1.

### FVP_VE_Cortex-A15x1 instances

**FVP_VE_Cortex_A15x1**

Top level component of the Cortex_A15x1 Versatile Express inspired model.
Type: `FVP_VE_Cortex_A15x1`.

**FVP_VE_Cortex_A15x1.cluster**

ARM Cortex-A15 Cluster CT model.
Type: `Cluster_ARM_Cortex-A15`.

**FVP_VE_Cortex_A15x1.cluster.cpu0**

ARM Cortex-A15 CT model.
Type: `ARM_Cortex-A15`.

**FVP_VE_Cortex_A15x1.cluster.cpu0.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_VE_Cortex_A15x1.cluster.cpu0.itlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_VE_Cortex_A15x1.cluster.cpu0.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A15x1.cluster.cpu0.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A15x1.cluster.l2_cache**

PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A15x1.daughterboard**

Daughtercard, inspired by the ARM Versatile Express development platform.
Type: `VEDaughterBoard`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A15x1.daughterboard.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x1.daughterboard.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x1.daughterboard.dmc**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A15x1.daughterboard.dmc_phy**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A15x1.daughterboard.dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A15x1.daughterboard.dram_aliased**

Allow TrustZone secure/normal bus signals to be routed separately.
Type: `TZSwitch`.

**FVP_VE_Cortex_A15x1.daughterboard.dram_limit_4**

Allow TrustZone secure/normal bus signals to be routed separately.
Type: `TZSwitch`.

**FVP_VE_Cortex_A15x1.daughterboard.dram_limit_8**

Allow TrustZone secure/normal bus signals to be routed separately.
Type: `TZSwitch`.

**FVP_VE_Cortex_A15x1.daughterboard.exclusive_monitor**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_VE_Cortex_A15x1.daughterboard.hdlcd**

ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).
Type: `PL370_HDLCD`.

**FVP_VE_Cortex_A15x1.daughterboard.hdlcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A15x1.daughterboard.hdlcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A15x1.daughterboard.hdlcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A15x1.daughterboard.hdlcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A15x1.daughterboard.introuter**

Interrupt Mapping peripheral (non-cascaded).
Type: `VEInterruptMapper`.

**FVP_VE_Cortex_A15x1.daughterboard.nonsecure_region**

Allow TrustZone secure/normal bus signals to be routed separately.
Type: `TZSwitch`.

**FVP_VE_Cortex_A15x1.daughterboard.secureDRAM**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A15x1.daughterboard.secureRO**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_VE_Cortex_A15x1.daughterboard.secureROloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_VE_Cortex_A15x1.daughterboard.secureSRAM**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A15x1.daughterboard.secure_region**

Allow TrustZone secure/normal bus signals to be routed separately.
Type: `TZSwitch`.

**FVP_VE_Cortex_A15x1.daughterboard.sram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A15x1.daughterboard.vedcc**
Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

**FVP_VE_Cortex_A15x1.globalcounter**
Memory Mapped Counter Module for Generic Timers.
Type: `MemoryMappedCounterModule`.

**FVP_VE_Cortex_A15x1.motherboard**
Model inspired by the ARM Versatile Express Motherboard.
Type: `VEMotherBoard`.

**FVP_VE_Cortex_A15x1.motherboard.Timer_0_1**
ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_VE_Cortex_A15x1.motherboard.Timer_0_1.clk_div0**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x1.motherboard.Timer_0_1.clk_div1**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x1.motherboard.Timer_0_1.counter0**
Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_VE_Cortex_A15x1.motherboard.Timer_0_1.counter1**
Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_VE_Cortex_A15x1.motherboard.Timer_2_3**
ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_VE_Cortex_A15x1.motherboard.Timer_2_3.clk_div0**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x1.motherboard.Timer_2_3.clk_div1**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x1.motherboard.Timer_2_3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_VE_Cortex_A15x1.motherboard.Timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_VE_Cortex_A15x1.motherboard.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**FVP_VE_Cortex_A15x1.motherboard.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x1.motherboard.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x1.motherboard.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x1.motherboard.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x1.motherboard.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A15x1.motherboard.dummy_local_dap_rom**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A15x1.motherboard.dummy_ram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A15x1.motherboard.dummy_usb**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A15x1.motherboard.flash0**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_VE_Cortex_A15x1.motherboard.flash1**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_VE_Cortex_A15x1.motherboard.flashloader0**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_VE_Cortex_A15x1.motherboard.flashloader1**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_VE_Cortex_A15x1.motherboard.hostbridge**
Host Socket Interface Component.
Type: `HostBridge`.

**FVP_VE_Cortex_A15x1.motherboard.mmc**
Generic Multimedia Card.
Type: `MMC`.

**FVP_VE_Cortex_A15x1.motherboard.mmc.timer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A15x1.motherboard.mmc.timer.timer**
A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A15x1.motherboard.mmc.timer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A15x1.motherboard.mmc.timer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A15x1.motherboard.pl011_uart0**
ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_A15x1.motherboard.pl011_uart0.clk_divider**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x1.motherboard.pl011_uart0.timer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A15x1.motherboard.pl011_uart0.timer.timer**
A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A15x1.motherboard.pl011_uart0.timer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A15x1.motherboard.pl011_uart0.timer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A15x1.motherboard.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_A15x1.motherboard.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x1.motherboard.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A15x1.motherboard.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A15x1.motherboard.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A15x1.motherboard.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A15x1.motherboard.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_A15x1.motherboard.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A15x1.motherboard.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A15x1.motherboard.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A15x1.motherboard.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A15x1.motherboard.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A15x1.motherboard.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_A15x1.motherboard.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x1.motherboard.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A15x1.motherboard.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A15x1.motherboard.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A15x1.motherboard.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A15x1.motherboard.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

**FVP_VE_Cortex_A15x1.motherboard.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A15x1.motherboard.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A15x1.motherboard.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A15x1.motherboard.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A15x1.motherboard.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_VE_Cortex_A15x1.motherboard.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A15x1.motherboard.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A15x1.motherboard.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A15x1.motherboard.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A15x1.motherboard.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_VE_Cortex_A15x1.motherboard.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x1.motherboard.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_VE_Cortex_A15x1.motherboard.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x1.motherboard.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_VE_Cortex_A15x1.motherboard.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

**FVP_VE_Cortex_A15x1.motherboard.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A15x1.motherboard.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A15x1.motherboard.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A15x1.motherboard.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A15x1.motherboard.pl180_mci**

ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

**FVP_VE_Cortex_A15x1.motherboard.ps2keyboard**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A15x1.motherboard.ps2keyboard.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A15x1.motherboard.ps2keyboard.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A15x1.motherboard.ps2keyboard.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A15x1.motherboard.ps2keyboard.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A15x1.motherboard.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

**FVP_VE_Cortex_A15x1.motherboard.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A15x1.motherboard.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A15x1.motherboard.ps2mouse.ps2_clocktimer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A15x1.motherboard.ps2mouse.ps2_clocktimer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A15x1.motherboard.psram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A15x1.motherboard.smsc_91c111**
SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_VE_Cortex_A15x1.motherboard.sp805_wdog**
ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_VE_Cortex_A15x1.motherboard.sp805_wdog.clocktimer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A15x1.motherboard.sp805_wdog.clocktimer.timer**
A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A15x1.motherboard.sp805_wdog.clocktimer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A15x1.motherboard.sp805_wdog.clocktimer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A15x1.motherboard.sp810_sysctrl**

Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_VE_Cortex_A15x1.motherboard.sp810_sysctrl.clkdiv_clk0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x1.motherboard.sp810_sysctrl.clkdiv_clk1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x1.motherboard.sp810_sysctrl.clkdiv_clk2**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x1.motherboard.sp810_sysctrl.clkdiv_clk3**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x1.motherboard.terminal_0**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_A15x1.motherboard.terminal_1**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_A15x1.motherboard.terminal_2**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_A15x1.motherboard.terminal_3**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_A15x1.motherboard.ve_sysregs**

Type: `VE_SysRegs`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A15x1.motherboard.virtioblockdevice**

virtio block device.
Type: `VirtioBlockDevice`.

**FVP_VE_Cortex_A15x1.motherboard.virtiop9device**

virtio P9 server.
Type: `VirtioP9Device`.

**FVP_VE_Cortex_A15x1.motherboard.vis**

Display window for VE using Visualisation library.
Type: `VEVisualisation`.

**FVP_VE_Cortex_A15x1.motherboard.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_VE_Cortex_A15x1.motherboard.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x1.motherboard.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A15x1.motherboard.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A15x1.motherboard.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A15x1.motherboard.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A15x1.motherboard.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x1.motherboard.vram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

# 15.2 FVP_VE_Cortex-A15x1-A7x1

List of instances in FVP_VE_Cortex-A15x1-A7x1.

## FVP_VE_Cortex-A15x1-A7x1 instances

**FVP_VE_Cortex_A15x1_A7x1**

Top level component of the Cortex A15x1 A7x1 Versatile Express inspired model.
Type: `FVP_VE_Cortex_A15x1_A7x1`.

**FVP_VE_Cortex_A15x1_A7x1.coretile**

Dual cluster ARM Cortex-A15x1 and ARM Cortex-A7x1 Core Tile.
Type: `ARM_Cortex_A15x1_A7x1_CT`.

**FVP_VE_Cortex_A15x1_A7x1.coretile.cci400**

Cache Coherent Interconnect for AXI4 ACE.
Type: `CCI400`.

**FVP_VE_Cortex_A15x1_A7x1.coretile.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x1_A7x1.coretile.clockdivider1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x1_A7x1.coretile.cluster0**

ARM Cortex-A15 Cluster CT model.
Type: `Cluster_ARM_Cortex-A15`.

**FVP_VE_Cortex_A15x1_A7x1.coretile.cluster0.cpu0**

ARM Cortex-A15 CT model.
Type: `ARM_Cortex-A15`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A15x1_A7x1.coretile.cluster0.cpu0.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_VE_Cortex_A15x1_A7x1.coretile.cluster0.cpu0.itlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_VE_Cortex_A15x1_A7x1.coretile.cluster0.cpu0.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A15x1_A7x1.coretile.cluster0.cpu0.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A15x1_A7x1.coretile.cluster0.l2_cache**
PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A15x1_A7x1.coretile.cluster1**
ARM Cortex-A7 Cluster CT model.
Type: `Cluster_ARM_Cortex-A7`.

**FVP_VE_Cortex_A15x1_A7x1.coretile.cluster1.cpu0**
ARM Cortex-A7 CT model.
Type: `ARM_Cortex-A7`.

**FVP_VE_Cortex_A15x1_A7x1.coretile.cluster1.cpu0.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_VE_Cortex_A15x1_A7x1.coretile.cluster1.cpu0.itlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_VE_Cortex_A15x1_A7x1.coretile.cluster1.cpu0.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A15x1_A7x1.coretile.cluster1.cpu0.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A15x1_A7x1.coretile.cluster1.l2_cache**
PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A15x1_A7x1.coretile.dualclustersystemconfigurationblock**
Dual Cluster System Configuration Block.
Type: `DualClusterSystemConfigurationBlock`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A15x1_A7x1.coretile.globalcounter**

Memory Mapped Counter Module for Generic Timers.
Type: `MemoryMappedCounterModule`.

**FVP_VE_Cortex_A15x1_A7x1.coretile.v7_vgic**

System VGIC architecture version v7.
Type: `v7_VGIC`.

**FVP_VE_Cortex_A15x1_A7x1.daughterboard**

Daughtercard, inspired by the ARM Versatile Express development platform.
Type: `VEDaughterBoard`.

**FVP_VE_Cortex_A15x1_A7x1.daughterboard.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x1_A7x1.daughterboard.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x1_A7x1.daughterboard.dmc**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A15x1_A7x1.daughterboard.dmc_phy**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A15x1_A7x1.daughterboard.dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A15x1_A7x1.daughterboard.dram_aliased**

Allow TrustZone secure/normal bus signals to be routed separately.
Type: `TZSwitch`.

**FVP_VE_Cortex_A15x1_A7x1.daughterboard.dram_limit_4**

Allow TrustZone secure/normal bus signals to be routed separately.
Type: `TZSwitch`.

**FVP_VE_Cortex_A15x1_A7x1.daughterboard.dram_limit_8**

Allow TrustZone secure/normal bus signals to be routed separately.
Type: `TZSwitch`.

**FVP_VE_Cortex_A15x1_A7x1.daughterboard.exclusive_monitor**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A15x1_A7x1.daughterboard.hdlcd**

ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).
Type: `PL370_HDLCD`.

**FVP_VE_Cortex_A15x1_A7x1.daughterboard.hdlcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A15x1_A7x1.daughterboard.hdlcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A15x1_A7x1.daughterboard.hdlcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A15x1_A7x1.daughterboard.hdlcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A15x1_A7x1.daughterboard.introuter**

Interrupt Mapping peripheral (non-cascaded).
Type: `VEInterruptMapper`.

**FVP_VE_Cortex_A15x1_A7x1.daughterboard.nonsecure_region**

Allow TrustZone secure/normal bus signals to be routed separately.
Type: `TZSwitch`.

**FVP_VE_Cortex_A15x1_A7x1.daughterboard.secureDRAM**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A15x1_A7x1.daughterboard.secureRO**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_VE_Cortex_A15x1_A7x1.daughterboard.secureROloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A15x1_A7x1.daughterboard.secureSRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A15x1_A7x1.daughterboard.secure_region**
Allow TrustZone secure/normal bus signals to be routed separately.
Type: `TZSwitch`.

**FVP_VE_Cortex_A15x1_A7x1.daughterboard.sram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A15x1_A7x1.daughterboard.vedcc**
Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard**
Model inspired by the ARM Versatile Express Motherboard.
Type: `VEMotherBoard`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.Timer_0_1**
ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.Timer_0_1.clk_div0**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.Timer_0_1.clk_div1**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.Timer_0_1.counter0**
Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.Timer_0_1.counter1**
Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.Timer_2_3**
ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.Timer_2_3.clk_div0**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.Timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.Timer_2_3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.Timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A15x1_A7x1.motherboard.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.dummy_local_dap_rom**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.dummy_ram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.dummy_usb**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.flash0**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.flash1**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.flashloader0**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.flashloader1**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.mmc**

Generic Multimedia Card.
Type: `MMC`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A15x1_A7x1.motherboard.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A15x1_A7x1.motherboard.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ClockTimerThread`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A15x1_A7x1.motherboard.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.pl111_clcd**
ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.pl111_clcd.pl11x_clcd**
Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.pl111_clcd.pl11x_clcd.timer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.pl111_clcd.pl11x_clcd.timer.timer**
A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.pl111_clcd.pl11x_clcd.timer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.pl111_clcd.pl11x_clcd.timer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.pl180_mci**
ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.ps2keyboard**
Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A15x1_A7x1.motherboard.ps2keyboard.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.ps2keyboard.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.ps2keyboard.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.ps2keyboard.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.ps2mouse.ps2_clocktimer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.ps2mouse.ps2_clocktimer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.psram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.smsc_91c111**
SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.sp805_wdog**
ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.sp805_wdog.clocktimer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.sp805_wdog.clocktimer.timer**
A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.sp805_wdog.clocktimer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.sp805_wdog.clocktimer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A15x1_A7x1.motherboard.sp810_sysctrl**

Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.sp810_sysctrl.clkdiv_clk0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.sp810_sysctrl.clkdiv_clk1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.sp810_sysctrl.clkdiv_clk2**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.sp810_sysctrl.clkdiv_clk3**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.terminal_0**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.terminal_1**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.terminal_2**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.terminal_3**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.ve_sysregs**

Type: `VE_SysRegs`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A15x1_A7x1.motherboard.virtioblockdevice**

virtio block device.
Type: `VirtioBlockDevice`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.virtiop9device**

virtio P9 server.
Type: `VirtioP9Device`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.vis**

Display window for VE using Visualisation library.
Type: `VEVisualisation`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A15x1_A7x1.motherboard.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x1_A7x1.motherboard.vram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

## 15.3 FVP_VE_Cortex-A15x2

List of instances in FVP_VE_Cortex-A15x2.

### FVP_VE_Cortex-A15x2 instances

**FVP_VE_Cortex_A15x2**

Top level component of the Cortex_A15x2 Versatile Express inspired model.
Type: `FVP_VE_Cortex_A15x2`.

**FVP_VE_Cortex_A15x2.cluster**

ARM Cortex-A15 Cluster CT model.
Type: `Cluster_ARM_Cortex-A15`.

**FVP_VE_Cortex_A15x2.cluster.cpu0**

ARM Cortex-A15 CT model.
Type: `ARM_Cortex-A15`.

**FVP_VE_Cortex_A15x2.cluster.cpu0.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_VE_Cortex_A15x2.cluster.cpu0.itlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_VE_Cortex_A15x2.cluster.cpu0.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A15x2.cluster.cpu0.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A15x2.cluster.cpu1**

ARM Cortex-A15 CT model.
Type: `ARM_Cortex-A15`.

**FVP_VE_Cortex_A15x2.cluster.cpu1.dtlb**

TLB - instruction, data or unified.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `TlbCadi`.

**FVP_VE_Cortex_A15x2.cluster.cpu1.itlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_VE_Cortex_A15x2.cluster.cpu1.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A15x2.cluster.cpu1.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A15x2.cluster.l2_cache**

PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A15x2.daughterboard**

Daughtercard, inspired by the ARM Versatile Express development platform.
Type: `VEDaughterBoard`.

**FVP_VE_Cortex_A15x2.daughterboard.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x2.daughterboard.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x2.daughterboard.dmc**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A15x2.daughterboard.dmc_phy**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A15x2.daughterboard.dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A15x2.daughterboard.dram_aliased**

Allow TrustZone secure/normal bus signals to be routed separately.
Type: `TZSwitch`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A15x2.daughterboard.dram_limit_4**

Allow TrustZone secure/normal bus signals to be routed separately.
Type: `TZSwitch`.

**FVP_VE_Cortex_A15x2.daughterboard.dram_limit_8**

Allow TrustZone secure/normal bus signals to be routed separately.
Type: `TZSwitch`.

**FVP_VE_Cortex_A15x2.daughterboard.exclusive_monitor**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_VE_Cortex_A15x2.daughterboard.hdlcd**

ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).
Type: `PL370_HDLCD`.

**FVP_VE_Cortex_A15x2.daughterboard.hdlcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A15x2.daughterboard.hdlcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A15x2.daughterboard.hdlcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A15x2.daughterboard.hdlcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A15x2.daughterboard.introuter**

Interrupt Mapping peripheral (non-cascaded).
Type: `VEInterruptMapper`.

**FVP_VE_Cortex_A15x2.daughterboard.nonsecure_region**

Allow TrustZone secure/normal bus signals to be routed separately.
Type: `TZSwitch`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A15x2.daughterboard.secureDRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A15x2.daughterboard.secureRO**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_VE_Cortex_A15x2.daughterboard.secureROloader**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_VE_Cortex_A15x2.daughterboard.secureSRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A15x2.daughterboard.secure_region**
Allow TrustZone secure/normal bus signals to be routed separately.
Type: `TZSwitch`.

**FVP_VE_Cortex_A15x2.daughterboard.sram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A15x2.daughterboard.vedcc**
Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

**FVP_VE_Cortex_A15x2.globalcounter**
Memory Mapped Counter Module for Generic Timers.
Type: `MemoryMappedCounterModule`.

**FVP_VE_Cortex_A15x2.motherboard**
Model inspired by the ARM Versatile Express Motherboard.
Type: `VEMotherBoard`.

**FVP_VE_Cortex_A15x2.motherboard.Timer_0_1**
ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_VE_Cortex_A15x2.motherboard.Timer_0_1.clk_div0**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x2.motherboard.Timer_0_1.clk_div1**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A15x2.motherboard.Timer_0_1.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_VE_Cortex_A15x2.motherboard.Timer_0_1.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_VE_Cortex_A15x2.motherboard.Timer_2_3**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_VE_Cortex_A15x2.motherboard.Timer_2_3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x2.motherboard.Timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x2.motherboard.Timer_2_3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_VE_Cortex_A15x2.motherboard.Timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_VE_Cortex_A15x2.motherboard.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**FVP_VE_Cortex_A15x2.motherboard.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x2.motherboard.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A15x2.motherboard.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x2.motherboard.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x2.motherboard.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x2.motherboard.dummy_local_dap_rom**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A15x2.motherboard.dummy_ram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A15x2.motherboard.dummy_usb**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A15x2.motherboard.flash0**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_VE_Cortex_A15x2.motherboard.flash1**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_VE_Cortex_A15x2.motherboard.flashloader0**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_VE_Cortex_A15x2.motherboard.flashloader1**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_VE_Cortex_A15x2.motherboard.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A15x2.motherboard.mmc**

Generic Multimedia Card.
Type: `MMC`.

**FVP_VE_Cortex_A15x2.motherboard.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A15x2.motherboard.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A15x2.motherboard.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A15x2.motherboard.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A15x2.motherboard.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_A15x2.motherboard.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x2.motherboard.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ClockTimerThread`.

**FVP_VE_Cortex_A15x2.motherboard.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A15x2.motherboard.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A15x2.motherboard.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A15x2.motherboard.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_A15x2.motherboard.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x2.motherboard.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A15x2.motherboard.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A15x2.motherboard.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A15x2.motherboard.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A15x2.motherboard.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_A15x2.motherboard.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x2.motherboard.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A15x2.motherboard.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A15x2.motherboard.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A15x2.motherboard.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A15x2.motherboard.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A15x2.motherboard.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x2.motherboard.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A15x2.motherboard.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A15x2.motherboard.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A15x2.motherboard.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A15x2.motherboard.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

**FVP_VE_Cortex_A15x2.motherboard.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A15x2.motherboard.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A15x2.motherboard.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A15x2.motherboard.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A15x2.motherboard.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_VE_Cortex_A15x2.motherboard.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A15x2.motherboard.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A15x2.motherboard.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A15x2.motherboard.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A15x2.motherboard.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_VE_Cortex_A15x2.motherboard.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x2.motherboard.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_VE_Cortex_A15x2.motherboard.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x2.motherboard.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_VE_Cortex_A15x2.motherboard.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

**FVP_VE_Cortex_A15x2.motherboard.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A15x2.motherboard.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A15x2.motherboard.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `SchedulerThread`.

**FVP_VE_Cortex_A15x2.motherboard.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A15x2.motherboard.pl180_mci**

ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

**FVP_VE_Cortex_A15x2.motherboard.ps2keyboard**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

**FVP_VE_Cortex_A15x2.motherboard.ps2keyboard.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A15x2.motherboard.ps2keyboard.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A15x2.motherboard.ps2keyboard.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A15x2.motherboard.ps2keyboard.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A15x2.motherboard.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A15x2.motherboard.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A15x2.motherboard.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A15x2.motherboard.ps2mouse.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A15x2.motherboard.ps2mouse.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A15x2.motherboard.psram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A15x2.motherboard.smsc_91c111**

SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_VE_Cortex_A15x2.motherboard.sp805_wdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_VE_Cortex_A15x2.motherboard.sp805_wdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A15x2.motherboard.sp805_wdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A15x2.motherboard.sp805_wdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A15x2.motherboard.sp805_wdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A15x2.motherboard.sp810_sysctrl**

Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_VE_Cortex_A15x2.motherboard.sp810_sysctrl.clkdiv_clk0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x2.motherboard.sp810_sysctrl.clkdiv_clk1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x2.motherboard.sp810_sysctrl.clkdiv_clk2**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x2.motherboard.sp810_sysctrl.clkdiv_clk3**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A15x2.motherboard.terminal_0**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_A15x2.motherboard.terminal_1**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_A15x2.motherboard.terminal_2**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_A15x2.motherboard.terminal_3**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_A15x2.motherboard.ve_sysregs**

Type: `VE_SysRegs`.

**FVP_VE_Cortex_A15x2.motherboard.virtioblockdevice**

virtio block device.
Type: `VirtioBlockDevice`.

**FVP_VE_Cortex_A15x2.motherboard.virtiop9device**

virtio P9 server.
Type: `VirtioP9Device`.

**FVP_VE_Cortex_A15x2.motherboard.vis**

Display window for VE using Visualisation library.
Type: `VEVisualisation`.

**FVP_VE_Cortex_A15x2.motherboard.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_VE_Cortex_A15x2.motherboard.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x2.motherboard.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A15x2.motherboard.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A15x2.motherboard.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A15x2.motherboard.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A15x2.motherboard.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x2.motherboard.vram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

# 15.4 FVP_VE_Cortex-A15x2-A7x2

List of instances in FVP_VE_Cortex-A15x2-A7x2.

## FVP_VE_Cortex-A15x2-A7x2 instances

**FVP_VE_Cortex_A15x2_A7x2**

Top level component of the Cortex A15x2 A7x2 Versatile Express inspired model.
Type: `FVP_VE_Cortex_A15x2_A7x2`.

**FVP_VE_Cortex_A15x2_A7x2.coretile**

Dual cluster ARM Cortex-A15x2 and ARM Cortex-A7x2 Core Tile.
Type: `ARM_Cortex_A15x2_A7x2_CT`.

**FVP_VE_Cortex_A15x2_A7x2.coretile.cci400**

Cache Coherent Interconnect for AXI4 ACE.
Type: `CCI400`.

**FVP_VE_Cortex_A15x2_A7x2.coretile.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x2_A7x2.coretile.clockdivider1**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x2_A7x2.coretile.cluster0**
ARM Cortex-A15 Cluster CT model.
Type: `Cluster_ARM_Cortex-A15`.

**FVP_VE_Cortex_A15x2_A7x2.coretile.cluster0.cpu0**
ARM Cortex-A15 CT model.
Type: `ARM_Cortex-A15`.

**FVP_VE_Cortex_A15x2_A7x2.coretile.cluster0.cpu0.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_VE_Cortex_A15x2_A7x2.coretile.cluster0.cpu0.itlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_VE_Cortex_A15x2_A7x2.coretile.cluster0.cpu0.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A15x2_A7x2.coretile.cluster0.cpu0.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A15x2_A7x2.coretile.cluster0.cpu1**
ARM Cortex-A15 CT model.
Type: `ARM_Cortex-A15`.

**FVP_VE_Cortex_A15x2_A7x2.coretile.cluster0.cpu1.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_VE_Cortex_A15x2_A7x2.coretile.cluster0.cpu1.itlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_VE_Cortex_A15x2_A7x2.coretile.cluster0.cpu1.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A15x2_A7x2.coretile.cluster0.cpu1.l1icache**
PV Cache.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `PVCache`.

**FVP_VE_Cortex_A15x2_A7x2.coretile.cluster0.l2_cache**

PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A15x2_A7x2.coretile.cluster1**

ARM Cortex-A7 Cluster CT model.
Type: `Cluster_ARM_Cortex-A7`.

**FVP_VE_Cortex_A15x2_A7x2.coretile.cluster1.cpu0**

ARM Cortex-A7 CT model.
Type: `ARM_Cortex-A7`.

**FVP_VE_Cortex_A15x2_A7x2.coretile.cluster1.cpu0.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_VE_Cortex_A15x2_A7x2.coretile.cluster1.cpu0.itlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_VE_Cortex_A15x2_A7x2.coretile.cluster1.cpu0.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A15x2_A7x2.coretile.cluster1.cpu0.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A15x2_A7x2.coretile.cluster1.cpu1**

ARM Cortex-A7 CT model.
Type: `ARM_Cortex-A7`.

**FVP_VE_Cortex_A15x2_A7x2.coretile.cluster1.cpu1.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_VE_Cortex_A15x2_A7x2.coretile.cluster1.cpu1.itlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_VE_Cortex_A15x2_A7x2.coretile.cluster1.cpu1.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A15x2_A7x2.coretile.cluster1.cpu1.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A15x2_A7x2.coretile.cluster1.l2_cache**

PV Cache.
Type: `PVCache`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A15x2_A7x2.coretile.dualclustersystemconfigurationblock**
Dual Cluster System Configuration Block.
Type: `DualClusterSystemConfigurationBlock`.

**FVP_VE_Cortex_A15x2_A7x2.coretile.globalcounter**
Memory Mapped Counter Module for Generic Timers.
Type: `MemoryMappedCounterModule`.

**FVP_VE_Cortex_A15x2_A7x2.coretile.v7_vgic**
System VGIC architecture version v7.
Type: `v7_VGIC`.

**FVP_VE_Cortex_A15x2_A7x2.daughterboard**
Daughtercard, inspired by the ARM Versatile Express development platform.
Type: `VEDaughterBoard`.

**FVP_VE_Cortex_A15x2_A7x2.daughterboard.clockCLCD**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x2_A7x2.daughterboard.clockdivider**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x2_A7x2.daughterboard.dmc**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A15x2_A7x2.daughterboard.dmc_phy**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A15x2_A7x2.daughterboard.dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A15x2_A7x2.daughterboard.dram_aliased**
Allow TrustZone secure/normal bus signals to be routed separately.
Type: `TZSwitch`.

**FVP_VE_Cortex_A15x2_A7x2.daughterboard.dram_limit_4**
Allow TrustZone secure/normal bus signals to be routed separately.
Type: `TZSwitch`.

**FVP_VE_Cortex_A15x2_A7x2.daughterboard.dram_limit_8**
Allow TrustZone secure/normal bus signals to be routed separately.
Type: `TZSwitch`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A15x2_A7x2.daughterboard.exclusive_monitor**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_VE_Cortex_A15x2_A7x2.daughterboard.hdlcd**

ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).
Type: `PL370_HDLCD`.

**FVP_VE_Cortex_A15x2_A7x2.daughterboard.hdlcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A15x2_A7x2.daughterboard.hdlcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A15x2_A7x2.daughterboard.hdlcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A15x2_A7x2.daughterboard.hdlcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A15x2_A7x2.daughterboard.introuter**

Interrupt Mapping peripheral (non-cascaded).
Type: `VEInterruptMapper`.

**FVP_VE_Cortex_A15x2_A7x2.daughterboard.nonsecure_region**

Allow TrustZone secure/normal bus signals to be routed separately.
Type: `TZSwitch`.

**FVP_VE_Cortex_A15x2_A7x2.daughterboard.secureDRAM**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A15x2_A7x2.daughterboard.secureRO**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A15x2_A7x2.daughterboard.secureROloader**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_VE_Cortex_A15x2_A7x2.daughterboard.secureSRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A15x2_A7x2.daughterboard.secure_region**
Allow TrustZone secure/normal bus signals to be routed separately.
Type: `TZSwitch`.

**FVP_VE_Cortex_A15x2_A7x2.daughterboard.sram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A15x2_A7x2.daughterboard.vedcc**
Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard**
Model inspired by the ARM Versatile Express Motherboard.
Type: `VEMotherBoard`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.Timer_0_1**
ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.Timer_0_1.clk_div0**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.Timer_0_1.clk_div1**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.Timer_0_1.counter0**
Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.Timer_0_1.counter1**
Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.Timer_2_3**
ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A15x2_A7x2.motherboard.Timer_2_3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.Timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.Timer_2_3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.Timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.dummy_local_dap_rom**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.dummy_ram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.dummy_usb**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.flash0**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.flash1**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.flashloader0**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.flashloader1**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.mmc**

Generic Multimedia Card.
Type: `MMC`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A15x2_A7x2.motherboard.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A15x2_A7x2.motherboard.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A15x2_A7x2.motherboard.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A15x2_A7x2.motherboard.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.pl180_mci**

ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.ps2keyboard**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `PS2Keyboard`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.ps2keyboard.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.ps2keyboard.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.ps2keyboard.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.ps2keyboard.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.ps2mouse.ps2_clocktimer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.ps2mouse.ps2_clocktimer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.psram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.smsc_91c111**
SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.sp805_wdog**
ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.sp805_wdog.clocktimer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.sp805_wdog.clocktimer.timer**
A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.sp805_wdog.clocktimer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.sp805_wdog.clocktimer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A15x2_A7x2.motherboard.sp810_sysctrl**

Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.sp810_sysctrl.clkdiv_clk0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.sp810_sysctrl.clkdiv_clk1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.sp810_sysctrl.clkdiv_clk2**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.sp810_sysctrl.clkdiv_clk3**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.terminal_0**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.terminal_1**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.terminal_2**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.terminal_3**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.ve_sysregs**

Type: `VE_SysRegs`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A15x2_A7x2.motherboard.virtioblockdevice**

virtio block device.
Type: `VirtioBlockDevice`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.virtiop9device**

virtio P9 server.
Type: `VirtioP9Device`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.vis**

Display window for VE using Visualisation library.
Type: `VEVisualisation`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A15x2_A7x2.motherboard.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x2_A7x2.motherboard.vram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

## 15.5 FVP_VE_Cortex-A15x4

List of instances in FVP_VE_Cortex-A15x4.

### FVP_VE_Cortex-A15x4 instances

**FVP_VE_Cortex_A15x4**

Top level component of the Cortex_A15x4 Versatile Express inspired model.
Type: `FVP_VE_Cortex_A15x4`.

**FVP_VE_Cortex_A15x4.cluster**

ARM Cortex-A15 Cluster CT model.
Type: `Cluster_ARM_Cortex-A15`.

**FVP_VE_Cortex_A15x4.cluster.cpu0**

ARM Cortex-A15 CT model.
Type: `ARM_Cortex-A15`.

**FVP_VE_Cortex_A15x4.cluster.cpu0.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_VE_Cortex_A15x4.cluster.cpu0.itlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_VE_Cortex_A15x4.cluster.cpu0.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A15x4.cluster.cpu0.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A15x4.cluster.cpu1**

ARM Cortex-A15 CT model.
Type: `ARM_Cortex-A15`.

**FVP_VE_Cortex_A15x4.cluster.cpu1.dtlb**

TLB - instruction, data or unified.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `TlbCadi`.

**FVP_VE_Cortex_A15x4.cluster.cpu1.itlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_VE_Cortex_A15x4.cluster.cpu1.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A15x4.cluster.cpu1.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A15x4.cluster.cpu2**
ARM Cortex-A15 CT model.
Type: `ARM_Cortex-A15`.

**FVP_VE_Cortex_A15x4.cluster.cpu2.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_VE_Cortex_A15x4.cluster.cpu2.itlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_VE_Cortex_A15x4.cluster.cpu2.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A15x4.cluster.cpu2.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A15x4.cluster.cpu3**
ARM Cortex-A15 CT model.
Type: `ARM_Cortex-A15`.

**FVP_VE_Cortex_A15x4.cluster.cpu3.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_VE_Cortex_A15x4.cluster.cpu3.itlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_VE_Cortex_A15x4.cluster.cpu3.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A15x4.cluster.cpu3.l1icache**
PV Cache.
Type: `PVCache`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A15x4.cluster.l2_cache**

PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A15x4.daughterboard**

Daughtercard, inspired by the ARM Versatile Express development platform.
Type: `VEDaughterBoard`.

**FVP_VE_Cortex_A15x4.daughterboard.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x4.daughterboard.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x4.daughterboard.dmc**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A15x4.daughterboard.dmc_phy**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A15x4.daughterboard.dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A15x4.daughterboard.dram_aliased**

Allow TrustZone secure/normal bus signals to be routed separately.
Type: `TZSwitch`.

**FVP_VE_Cortex_A15x4.daughterboard.dram_limit_4**

Allow TrustZone secure/normal bus signals to be routed separately.
Type: `TZSwitch`.

**FVP_VE_Cortex_A15x4.daughterboard.dram_limit_8**

Allow TrustZone secure/normal bus signals to be routed separately.
Type: `TZSwitch`.

**FVP_VE_Cortex_A15x4.daughterboard.exclusive_monitor**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_VE_Cortex_A15x4.daughterboard.hdlcd**

ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).
Type: `PL370_HDLCD`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A15x4.daughterboard.hdlcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A15x4.daughterboard.hdlcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A15x4.daughterboard.hdlcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A15x4.daughterboard.hdlcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A15x4.daughterboard.introuter**

Interrupt Mapping peripheral (non-cascaded).
Type: `VEInterruptMapper`.

**FVP_VE_Cortex_A15x4.daughterboard.nonsecure_region**

Allow TrustZone secure/normal bus signals to be routed separately.
Type: `TZSwitch`.

**FVP_VE_Cortex_A15x4.daughterboard.secureDRAM**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A15x4.daughterboard.secureRO**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_VE_Cortex_A15x4.daughterboard.secureROloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_VE_Cortex_A15x4.daughterboard.secureSRAM**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A15x4.daughterboard.secure_region**
Allow TrustZone secure/normal bus signals to be routed separately.
Type: `TZSwitch`.

**FVP_VE_Cortex_A15x4.daughterboard.sram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A15x4.daughterboard.vedcc**
Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

**FVP_VE_Cortex_A15x4.globalcounter**
Memory Mapped Counter Module for Generic Timers.
Type: `MemoryMappedCounterModule`.

**FVP_VE_Cortex_A15x4.motherboard**
Model inspired by the ARM Versatile Express Motherboard.
Type: `VEMotherBoard`.

**FVP_VE_Cortex_A15x4.motherboard.Timer_0_1**
ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_VE_Cortex_A15x4.motherboard.Timer_0_1.clk_div0**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x4.motherboard.Timer_0_1.clk_div1**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x4.motherboard.Timer_0_1.counter0**
Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_VE_Cortex_A15x4.motherboard.Timer_0_1.counter1**
Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_VE_Cortex_A15x4.motherboard.Timer_2_3**
ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_VE_Cortex_A15x4.motherboard.Timer_2_3.clk_div0**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x4.motherboard.Timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x4.motherboard.Timer_2_3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_VE_Cortex_A15x4.motherboard.Timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_VE_Cortex_A15x4.motherboard.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**FVP_VE_Cortex_A15x4.motherboard.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x4.motherboard.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x4.motherboard.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x4.motherboard.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A15x4.motherboard.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x4.motherboard.dummy_local_dap_rom**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A15x4.motherboard.dummy_ram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A15x4.motherboard.dummy_usb**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A15x4.motherboard.flash0**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_VE_Cortex_A15x4.motherboard.flash1**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_VE_Cortex_A15x4.motherboard.flashloader0**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_VE_Cortex_A15x4.motherboard.flashloader1**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_VE_Cortex_A15x4.motherboard.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_VE_Cortex_A15x4.motherboard.mmc**

Generic Multimedia Card.
Type: `MMC`.

**FVP_VE_Cortex_A15x4.motherboard.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A15x4.motherboard.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A15x4.motherboard.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A15x4.motherboard.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A15x4.motherboard.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_A15x4.motherboard.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x4.motherboard.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A15x4.motherboard.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A15x4.motherboard.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A15x4.motherboard.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A15x4.motherboard.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_A15x4.motherboard.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x4.motherboard.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A15x4.motherboard.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A15x4.motherboard.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A15x4.motherboard.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A15x4.motherboard.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_A15x4.motherboard.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x4.motherboard.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A15x4.motherboard.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A15x4.motherboard.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A15x4.motherboard.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A15x4.motherboard.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_A15x4.motherboard.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x4.motherboard.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ClockTimerThread`.

**FVP_VE_Cortex_A15x4.motherboard.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A15x4.motherboard.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A15x4.motherboard.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A15x4.motherboard.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

**FVP_VE_Cortex_A15x4.motherboard.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A15x4.motherboard.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A15x4.motherboard.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A15x4.motherboard.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A15x4.motherboard.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_VE_Cortex_A15x4.motherboard.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A15x4.motherboard.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A15x4.motherboard.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A15x4.motherboard.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A15x4.motherboard.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_VE_Cortex_A15x4.motherboard.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x4.motherboard.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_VE_Cortex_A15x4.motherboard.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x4.motherboard.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_VE_Cortex_A15x4.motherboard.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

**FVP_VE_Cortex_A15x4.motherboard.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A15x4.motherboard.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A15x4.motherboard.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A15x4.motherboard.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A15x4.motherboard.pl180_mci**

ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

**FVP_VE_Cortex_A15x4.motherboard.ps2keyboard**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A15x4.motherboard.ps2keyboard.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A15x4.motherboard.ps2keyboard.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A15x4.motherboard.ps2keyboard.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A15x4.motherboard.ps2keyboard.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A15x4.motherboard.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

**FVP_VE_Cortex_A15x4.motherboard.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A15x4.motherboard.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A15x4.motherboard.ps2mouse.ps2_clocktimer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A15x4.motherboard.ps2mouse.ps2_clocktimer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A15x4.motherboard.psram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A15x4.motherboard.smsc_91c111**
SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_VE_Cortex_A15x4.motherboard.sp805_wdog**
ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_VE_Cortex_A15x4.motherboard.sp805_wdog.clocktimer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A15x4.motherboard.sp805_wdog.clocktimer.timer**
A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A15x4.motherboard.sp805_wdog.clocktimer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A15x4.motherboard.sp805_wdog.clocktimer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A15x4.motherboard.sp810_sysctrl**

Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_VE_Cortex_A15x4.motherboard.sp810_sysctrl.clkdiv_clk0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x4.motherboard.sp810_sysctrl.clkdiv_clk1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x4.motherboard.sp810_sysctrl.clkdiv_clk2**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x4.motherboard.sp810_sysctrl.clkdiv_clk3**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x4.motherboard.terminal_0**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_A15x4.motherboard.terminal_1**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_A15x4.motherboard.terminal_2**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_A15x4.motherboard.terminal_3**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_A15x4.motherboard.ve_sysregs**

Type: `VE_SysRegs`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A15x4.motherboard.virtioblockdevice**

virtio block device.
Type: `VirtioBlockDevice`.

**FVP_VE_Cortex_A15x4.motherboard.virtiop9device**

virtio P9 server.
Type: `VirtioP9Device`.

**FVP_VE_Cortex_A15x4.motherboard.vis**

Display window for VE using Visualisation library.
Type: `VEVisualisation`.

**FVP_VE_Cortex_A15x4.motherboard.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_VE_Cortex_A15x4.motherboard.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x4.motherboard.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A15x4.motherboard.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A15x4.motherboard.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A15x4.motherboard.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A15x4.motherboard.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x4.motherboard.vram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

## 15.6 FVP_VE_Cortex-A15x4-A7x4

List of instances in FVP_VE_Cortex-A15x4-A7x4.

### FVP_VE_Cortex-A15x4-A7x4 instances

**FVP_VE_Cortex_A15x4_A7x4**

Top level component of the Cortex A15x4 A7x4 Versatile Express inspired model.
Type: `FVP_VE_Cortex_A15x4_A7x4`.

**FVP_VE_Cortex_A15x4_A7x4.coretile**

Dual cluster ARM Cortex-A15x4 and ARM Cortex-A7x4 Core Tile.
Type: `ARM_Cortex_A15x4_A7x4_CT`.

**FVP_VE_Cortex_A15x4_A7x4.coretile.cci400**

Cache Coherent Interconnect for AXI4 ACE.
Type: `CCI400`.

**FVP_VE_Cortex_A15x4_A7x4.coretile.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x4_A7x4.coretile.clockdivider1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x4_A7x4.coretile.cluster0**

ARM Cortex-A15 Cluster CT model.
Type: `Cluster_ARM_Cortex-A15`.

**FVP_VE_Cortex_A15x4_A7x4.coretile.cluster0.cpu0**

ARM Cortex-A15 CT model.
Type: `ARM_Cortex-A15`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A15x4_A7x4.coretile.cluster0.cpu0.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_VE_Cortex_A15x4_A7x4.coretile.cluster0.cpu0.itlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_VE_Cortex_A15x4_A7x4.coretile.cluster0.cpu0.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A15x4_A7x4.coretile.cluster0.cpu0.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A15x4_A7x4.coretile.cluster0.cpu1**
ARM Cortex-A15 CT model.
Type: `ARM_Cortex-A15`.

**FVP_VE_Cortex_A15x4_A7x4.coretile.cluster0.cpu1.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_VE_Cortex_A15x4_A7x4.coretile.cluster0.cpu1.itlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_VE_Cortex_A15x4_A7x4.coretile.cluster0.cpu1.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A15x4_A7x4.coretile.cluster0.cpu1.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A15x4_A7x4.coretile.cluster0.cpu2**
ARM Cortex-A15 CT model.
Type: `ARM_Cortex-A15`.

**FVP_VE_Cortex_A15x4_A7x4.coretile.cluster0.cpu2.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_VE_Cortex_A15x4_A7x4.coretile.cluster0.cpu2.itlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_VE_Cortex_A15x4_A7x4.coretile.cluster0.cpu2.l1dcache**
PV Cache.
Type: `PVCache`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A15x4_A7x4.coretile.cluster0.cpu2.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A15x4_A7x4.coretile.cluster0.cpu3**
ARM Cortex-A15 CT model.
Type: `ARM_Cortex-A15`.

**FVP_VE_Cortex_A15x4_A7x4.coretile.cluster0.cpu3.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_VE_Cortex_A15x4_A7x4.coretile.cluster0.cpu3.itlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_VE_Cortex_A15x4_A7x4.coretile.cluster0.cpu3.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A15x4_A7x4.coretile.cluster0.cpu3.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A15x4_A7x4.coretile.cluster0.l2_cache**
PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A15x4_A7x4.coretile.cluster1**
ARM Cortex-A7 Cluster CT model.
Type: `Cluster_ARM_Cortex-A7`.

**FVP_VE_Cortex_A15x4_A7x4.coretile.cluster1.cpu0**
ARM Cortex-A7 CT model.
Type: `ARM_Cortex-A7`.

**FVP_VE_Cortex_A15x4_A7x4.coretile.cluster1.cpu0.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_VE_Cortex_A15x4_A7x4.coretile.cluster1.cpu0.itlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_VE_Cortex_A15x4_A7x4.coretile.cluster1.cpu0.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A15x4_A7x4.coretile.cluster1.cpu0.l1icache**
PV Cache.
Type: `PVCache`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A15x4_A7x4.coretile.cluster1.cpu1**
ARM Cortex-A7 CT model.
Type: `ARM_Cortex-A7`.

**FVP_VE_Cortex_A15x4_A7x4.coretile.cluster1.cpu1.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_VE_Cortex_A15x4_A7x4.coretile.cluster1.cpu1.itlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_VE_Cortex_A15x4_A7x4.coretile.cluster1.cpu1.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A15x4_A7x4.coretile.cluster1.cpu1.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A15x4_A7x4.coretile.cluster1.cpu2**
ARM Cortex-A7 CT model.
Type: `ARM_Cortex-A7`.

**FVP_VE_Cortex_A15x4_A7x4.coretile.cluster1.cpu2.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_VE_Cortex_A15x4_A7x4.coretile.cluster1.cpu2.itlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_VE_Cortex_A15x4_A7x4.coretile.cluster1.cpu2.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A15x4_A7x4.coretile.cluster1.cpu2.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A15x4_A7x4.coretile.cluster1.cpu3**
ARM Cortex-A7 CT model.
Type: `ARM_Cortex-A7`.

**FVP_VE_Cortex_A15x4_A7x4.coretile.cluster1.cpu3.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_VE_Cortex_A15x4_A7x4.coretile.cluster1.cpu3.itlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A15x4_A7x4.coretile.cluster1.cpu3.l1dcache**
PV Cache.
Type: PVCache.

**FVP_VE_Cortex_A15x4_A7x4.coretile.cluster1.cpu3.l1icache**
PV Cache.
Type: PVCache.

**FVP_VE_Cortex_A15x4_A7x4.coretile.cluster1.l2_cache**
PV Cache.
Type: PVCache.

**FVP_VE_Cortex_A15x4_A7x4.coretile.dualclustersystemconfigurationblock**
Dual Cluster System Configuration Block.
Type: DualClusterSystemConfigurationBlock.

**FVP_VE_Cortex_A15x4_A7x4.coretile.globalcounter**
Memory Mapped Counter Module for Generic Timers.
Type: MemoryMappedCounterModule.

**FVP_VE_Cortex_A15x4_A7x4.coretile.v7_vgic**
System VGIC architecture version v7.
Type: v7_VGIC.

**FVP_VE_Cortex_A15x4_A7x4.daughterboard**
Daughtercard, inspired by the ARM Versatile Express development platform.
Type: VEDaughterBoard.

**FVP_VE_Cortex_A15x4_A7x4.daughterboard.clockCLCD**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: ClockDivider.

**FVP_VE_Cortex_A15x4_A7x4.daughterboard.clockdivider**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: ClockDivider.

**FVP_VE_Cortex_A15x4_A7x4.daughterboard.dmc**
RAM device, can be dynamic or static ram.
Type: RAMDevice.

**FVP_VE_Cortex_A15x4_A7x4.daughterboard.dmc_phy**
RAM device, can be dynamic or static ram.
Type: RAMDevice.

**FVP_VE_Cortex_A15x4_A7x4.daughterboard.dram**
RAM device, can be dynamic or static ram.
Type: RAMDevice.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A15x4_A7x4.daughterboard.dram_aliased**

Allow TrustZone secure/normal bus signals to be routed separately.
Type: `TZSwitch`.

**FVP_VE_Cortex_A15x4_A7x4.daughterboard.dram_limit_4**

Allow TrustZone secure/normal bus signals to be routed separately.
Type: `TZSwitch`.

**FVP_VE_Cortex_A15x4_A7x4.daughterboard.dram_limit_8**

Allow TrustZone secure/normal bus signals to be routed separately.
Type: `TZSwitch`.

**FVP_VE_Cortex_A15x4_A7x4.daughterboard.exclusive_monitor**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_VE_Cortex_A15x4_A7x4.daughterboard.hdlcd**

ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).
Type: `PL370_HDLCD`.

**FVP_VE_Cortex_A15x4_A7x4.daughterboard.hdlcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A15x4_A7x4.daughterboard.hdlcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A15x4_A7x4.daughterboard.hdlcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A15x4_A7x4.daughterboard.hdlcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A15x4_A7x4.daughterboard.introuter**

Interrupt Mapping peripheral (non-cascaded).
Type: `VEInterruptMapper`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A15x4_A7x4.daughterboard.nonsecure_region**
Allow TrustZone secure/normal bus signals to be routed separately.
Type: `TZSwitch`.

**FVP_VE_Cortex_A15x4_A7x4.daughterboard.secureDRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A15x4_A7x4.daughterboard.secureRO**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_VE_Cortex_A15x4_A7x4.daughterboard.secureROloader**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_VE_Cortex_A15x4_A7x4.daughterboard.secureSRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A15x4_A7x4.daughterboard.secure_region**
Allow TrustZone secure/normal bus signals to be routed separately.
Type: `TZSwitch`.

**FVP_VE_Cortex_A15x4_A7x4.daughterboard.sram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A15x4_A7x4.daughterboard.vedcc**
Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard**
Model inspired by the ARM Versatile Express Motherboard.
Type: `VEMotherBoard`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.Timer_0_1**
ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.Timer_0_1.clk_div0**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.Timer_0_1.clk_div1**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A15x4_A7x4.motherboard.Timer_0_1.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.Timer_0_1.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.Timer_2_3**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.Timer_2_3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.Timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.Timer_2_3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.Timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A15x4_A7x4.motherboard.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.dummy_local_dap_rom**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.dummy_ram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.dummy_usb**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.flash0**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.flash1**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.flashloader0**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.flashloader1**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A15x4_A7x4.motherboard.mmc**

Generic Multimedia Card.
Type: `MMC`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ClockTimerThread`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A15x4_A7x4.motherboard.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A15x4_A7x4.motherboard.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A15x4_A7x4.motherboard.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A15x4_A7x4.motherboard.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `SchedulerThread`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.pl180_mci**

ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.ps2keyboard**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.ps2keyboard.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.ps2keyboard.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.ps2keyboard.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.ps2keyboard.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A15x4_A7x4.motherboard.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.ps2mouse.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.ps2mouse.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.psram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.smsc_91c111**

SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.sp805_wdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.sp805_wdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A15x4_A7x4.motherboard.sp805_wdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.sp805_wdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.sp805_wdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.sp810_sysctrl**

Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.sp810_sysctrl.clkdiv_clk0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.sp810_sysctrl.clkdiv_clk1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.sp810_sysctrl.clkdiv_clk2**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.sp810_sysctrl.clkdiv_clk3**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A15x4_A7x4.motherboard.terminal_0**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.terminal_1**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.terminal_2**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.terminal_3**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.ve_sysregs**
Type: `VE_SysRegs`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.virtioblockdevice**
virtio block device.
Type: `VirtioBlockDevice`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.virtiop9device**
virtio P9 server.
Type: `VirtioP9Device`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.vis**
Display window for VE using Visualisation library.
Type: `VEVisualisation`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.vis.recorder**
Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.vis.recorder.playbackDivider**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.vis.recorder.playbackTimer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A15x4_A7x4.motherboard.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A15x4_A7x4.motherboard.vram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

# 15.7 FVP_VE_Cortex-A17x1

List of instances in FVP_VE_Cortex-A17x1.

## FVP_VE_Cortex-A17x1 instances

**FVP_VE_Cortex_A17x1**

Top level component of the Cortex_A17x1 Versatile Express inspired model.
Type: `FVP_VE_Cortex_A17x1`.

**FVP_VE_Cortex_A17x1.cluster**

ARM Cortex-A17 Cluster CT model.
Type: `Cluster_ARM_Cortex-A17`.

**FVP_VE_Cortex_A17x1.cluster.cpu0**

ARM Cortex-A17 CT model.
Type: `ARM_Cortex-A17`.

**FVP_VE_Cortex_A17x1.cluster.cpu0.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A17x1.cluster.cpu0.itlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_VE_Cortex_A17x1.cluster.cpu0.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A17x1.cluster.cpu0.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A17x1.cluster.l2_cache**

PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A17x1.daughterboard**

Daughtercard, inspired by the ARM Versatile Express development platform.
Type: `VEDaughterBoard`.

**FVP_VE_Cortex_A17x1.daughterboard.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A17x1.daughterboard.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A17x1.daughterboard.dmc**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A17x1.daughterboard.dmc_phy**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A17x1.daughterboard.dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A17x1.daughterboard.dram_aliased**

Allow TrustZone secure/normal bus signals to be routed separately.
Type: `TZSwitch`.

**FVP_VE_Cortex_A17x1.daughterboard.dram_limit_4**

Allow TrustZone secure/normal bus signals to be routed separately.
Type: `TZSwitch`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A17x1.daughterboard.dram_limit_8**

Allow TrustZone secure/normal bus signals to be routed separately.
Type: `TZSwitch`.

**FVP_VE_Cortex_A17x1.daughterboard.exclusive_monitor**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_VE_Cortex_A17x1.daughterboard.hdlcd**

ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).
Type: `PL370_HDLCD`.

**FVP_VE_Cortex_A17x1.daughterboard.hdlcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A17x1.daughterboard.hdlcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A17x1.daughterboard.hdlcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A17x1.daughterboard.hdlcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A17x1.daughterboard.introuter**

Interrupt Mapping peripheral (non-cascaded).
Type: `VEInterruptMapper`.

**FVP_VE_Cortex_A17x1.daughterboard.nonsecure_region**

Allow TrustZone secure/normal bus signals to be routed separately.
Type: `TZSwitch`.

**FVP_VE_Cortex_A17x1.daughterboard.secureDRAM**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A17x1.daughterboard.secureRO**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_VE_Cortex_A17x1.daughterboard.secureROloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_VE_Cortex_A17x1.daughterboard.secureSRAM**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A17x1.daughterboard.secure_region**

Allow TrustZone secure/normal bus signals to be routed separately.
Type: `TZSwitch`.

**FVP_VE_Cortex_A17x1.daughterboard.sram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A17x1.daughterboard.vedcc**

Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

**FVP_VE_Cortex_A17x1.gic400**

GIC-400 Generic Interrupt Controller.
Type: `GIC_400`.

**FVP_VE_Cortex_A17x1.globalcounter**

Memory Mapped Counter Module for Generic Timers.
Type: `MemoryMappedCounterModule`.

**FVP_VE_Cortex_A17x1.motherboard**

Model inspired by the ARM Versatile Express Motherboard.
Type: `VEMotherBoard`.

**FVP_VE_Cortex_A17x1.motherboard.Timer_0_1**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_VE_Cortex_A17x1.motherboard.Timer_0_1.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A17x1.motherboard.Timer_0_1.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A17x1.motherboard.Timer_0_1.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_VE_Cortex_A17x1.motherboard.Timer_0_1.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_VE_Cortex_A17x1.motherboard.Timer_2_3**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_VE_Cortex_A17x1.motherboard.Timer_2_3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A17x1.motherboard.Timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A17x1.motherboard.Timer_2_3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_VE_Cortex_A17x1.motherboard.Timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_VE_Cortex_A17x1.motherboard.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**FVP_VE_Cortex_A17x1.motherboard.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A17x1.motherboard.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A17x1.motherboard.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A17x1.motherboard.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A17x1.motherboard.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A17x1.motherboard.dummy_local_dap_rom**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A17x1.motherboard.dummy_ram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A17x1.motherboard.dummy_usb**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A17x1.motherboard.flash0**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_VE_Cortex_A17x1.motherboard.flash1**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_VE_Cortex_A17x1.motherboard.flashloader0**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_VE_Cortex_A17x1.motherboard.flashloader1**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_VE_Cortex_A17x1.motherboard.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A17x1.motherboard.mmc**

Generic Multimedia Card.
Type: `MMC`.

**FVP_VE_Cortex_A17x1.motherboard.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A17x1.motherboard.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A17x1.motherboard.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A17x1.motherboard.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A17x1.motherboard.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_A17x1.motherboard.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A17x1.motherboard.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ClockTimerThread`.

**FVP_VE_Cortex_A17x1.motherboard.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A17x1.motherboard.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A17x1.motherboard.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A17x1.motherboard.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_A17x1.motherboard.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A17x1.motherboard.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A17x1.motherboard.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A17x1.motherboard.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A17x1.motherboard.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A17x1.motherboard.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_A17x1.motherboard.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A17x1.motherboard.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A17x1.motherboard.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A17x1.motherboard.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A17x1.motherboard.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A17x1.motherboard.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A17x1.motherboard.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A17x1.motherboard.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A17x1.motherboard.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A17x1.motherboard.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A17x1.motherboard.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A17x1.motherboard.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

**FVP_VE_Cortex_A17x1.motherboard.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A17x1.motherboard.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A17x1.motherboard.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A17x1.motherboard.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A17x1.motherboard.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_VE_Cortex_A17x1.motherboard.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A17x1.motherboard.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A17x1.motherboard.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A17x1.motherboard.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A17x1.motherboard.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_VE_Cortex_A17x1.motherboard.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A17x1.motherboard.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_VE_Cortex_A17x1.motherboard.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A17x1.motherboard.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_VE_Cortex_A17x1.motherboard.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

**FVP_VE_Cortex_A17x1.motherboard.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A17x1.motherboard.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A17x1.motherboard.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `SchedulerThread`.

**FVP_VE_Cortex_A17x1.motherboard.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A17x1.motherboard.pl180_mci**

ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

**FVP_VE_Cortex_A17x1.motherboard.ps2keyboard**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

**FVP_VE_Cortex_A17x1.motherboard.ps2keyboard.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A17x1.motherboard.ps2keyboard.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A17x1.motherboard.ps2keyboard.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A17x1.motherboard.ps2keyboard.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A17x1.motherboard.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A17x1.motherboard.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A17x1.motherboard.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A17x1.motherboard.ps2mouse.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A17x1.motherboard.ps2mouse.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A17x1.motherboard.psram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A17x1.motherboard.smsc_91c111**

SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_VE_Cortex_A17x1.motherboard.sp805_wdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_VE_Cortex_A17x1.motherboard.sp805_wdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A17x1.motherboard.sp805_wdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A17x1.motherboard.sp805_wdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A17x1.motherboard.sp805_wdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A17x1.motherboard.sp810_sysctrl**

Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_VE_Cortex_A17x1.motherboard.sp810_sysctrl.clkdiv_clk0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A17x1.motherboard.sp810_sysctrl.clkdiv_clk1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A17x1.motherboard.sp810_sysctrl.clkdiv_clk2**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A17x1.motherboard.sp810_sysctrl.clkdiv_clk3**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A17x1.motherboard.terminal_0**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_A17x1.motherboard.terminal_1**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_A17x1.motherboard.terminal_2**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_A17x1.motherboard.terminal_3**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_A17x1.motherboard.ve_sysregs**
Type: `VE_SysRegs`.

**FVP_VE_Cortex_A17x1.motherboard.virtioblockdevice**
virtio block device.
Type: `VirtioBlockDevice`.

**FVP_VE_Cortex_A17x1.motherboard.virtiop9device**
virtio P9 server.
Type: `VirtioP9Device`.

**FVP_VE_Cortex_A17x1.motherboard.vis**
Display window for VE using Visualisation library.
Type: `VEVisualisation`.

**FVP_VE_Cortex_A17x1.motherboard.vis.recorder**
Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_VE_Cortex_A17x1.motherboard.vis.recorder.playbackDivider**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A17x1.motherboard.vis.recorder.playbackTimer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A17x1.motherboard.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A17x1.motherboard.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A17x1.motherboard.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A17x1.motherboard.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A17x1.motherboard.vram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

## 15.8 FVP_VE_Cortex-A17x1-A7x1

List of instances in FVP_VE_Cortex-A17x1-A7x1.

### FVP_VE_Cortex-A17x1-A7x1 instances

**FVP_VE_Cortex_A17x1_A7x1**

Top level component of the Cortex A17x1 A7x1 Versatile Express inspired model.
Type: `FVP_VE_Cortex_A17x1_A7x1`.

**FVP_VE_Cortex_A17x1_A7x1.coretile**

Dual cluster ARM Cortex-A17x1 and ARM Cortex-A7x1 Core Tile.
Type: `ARM_Cortex_A17x1_A7x1_CT`.

**FVP_VE_Cortex_A17x1_A7x1.coretile.cci400**

Cache Coherent Interconnect for AXI4 ACE.
Type: `CCI400`.

**FVP_VE_Cortex_A17x1_A7x1.coretile.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A17x1_A7x1.coretile.clockdivider1**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A17x1_A7x1.coretile.cluster0**
ARM Cortex-A17 Cluster CT model.
Type: `Cluster_ARM_Cortex-A17`.

**FVP_VE_Cortex_A17x1_A7x1.coretile.cluster0.cpu0**
ARM Cortex-A17 CT model.
Type: `ARM_Cortex-A17`.

**FVP_VE_Cortex_A17x1_A7x1.coretile.cluster0.cpu0.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_VE_Cortex_A17x1_A7x1.coretile.cluster0.cpu0.itlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_VE_Cortex_A17x1_A7x1.coretile.cluster0.cpu0.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A17x1_A7x1.coretile.cluster0.cpu0.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A17x1_A7x1.coretile.cluster0.l2_cache**
PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A17x1_A7x1.coretile.cluster1**
ARM Cortex-A7 Cluster CT model.
Type: `Cluster_ARM_Cortex-A7`.

**FVP_VE_Cortex_A17x1_A7x1.coretile.cluster1.cpu0**
ARM Cortex-A7 CT model.
Type: `ARM_Cortex-A7`.

**FVP_VE_Cortex_A17x1_A7x1.coretile.cluster1.cpu0.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_VE_Cortex_A17x1_A7x1.coretile.cluster1.cpu0.itlb**
TLB - instruction, data or unified.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `TlbCadi`.

**FVP_VE_Cortex_A17x1_A7x1.coretile.cluster1.cpu0.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A17x1_A7x1.coretile.cluster1.cpu0.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A17x1_A7x1.coretile.cluster1.l2_cache**

PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A17x1_A7x1.coretile.dualclustersystemconfigurationblock**

Dual Cluster System Configuration Block.
Type: `DualClusterSystemConfigurationBlock`.

**FVP_VE_Cortex_A17x1_A7x1.coretile.globalcounter**

Memory Mapped Counter Module for Generic Timers.
Type: `MemoryMappedCounterModule`.

**FVP_VE_Cortex_A17x1_A7x1.coretile.v7_vgic**

System VGIC architecture version v7.
Type: `v7_VGIC`.

**FVP_VE_Cortex_A17x1_A7x1.daughterboard**

Daughtercard, inspired by the ARM Versatile Express development platform.
Type: `VEDaughterBoard`.

**FVP_VE_Cortex_A17x1_A7x1.daughterboard.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A17x1_A7x1.daughterboard.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A17x1_A7x1.daughterboard.dmc**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A17x1_A7x1.daughterboard.dmc_phy**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A17x1_A7x1.daughterboard.dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A17x1_A7x1.daughterboard.dram_aliased**
Allow TrustZone secure/normal bus signals to be routed separately.
Type: `TZSwitch`.

**FVP_VE_Cortex_A17x1_A7x1.daughterboard.dram_limit_4**
Allow TrustZone secure/normal bus signals to be routed separately.
Type: `TZSwitch`.

**FVP_VE_Cortex_A17x1_A7x1.daughterboard.dram_limit_8**
Allow TrustZone secure/normal bus signals to be routed separately.
Type: `TZSwitch`.

**FVP_VE_Cortex_A17x1_A7x1.daughterboard.exclusive_monitor**
Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_VE_Cortex_A17x1_A7x1.daughterboard.hdlcd**
ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).
Type: `PL370_HDLCD`.

**FVP_VE_Cortex_A17x1_A7x1.daughterboard.hdlcd.timer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A17x1_A7x1.daughterboard.hdlcd.timer.timer**
A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A17x1_A7x1.daughterboard.hdlcd.timer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A17x1_A7x1.daughterboard.hdlcd.timer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A17x1_A7x1.daughterboard.introuter**
Interrupt Mapping peripheral (non-cascaded).
Type: `VEInterruptMapper`.

**FVP_VE_Cortex_A17x1_A7x1.daughterboard.nonsecure_region**
Allow TrustZone secure/normal bus signals to be routed separately.
Type: `TZSwitch`.

**FVP_VE_Cortex_A17x1_A7x1.daughterboard.secureDRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A17x1_A7x1.daughterboard.secureRO**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_VE_Cortex_A17x1_A7x1.daughterboard.secureROloader**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_VE_Cortex_A17x1_A7x1.daughterboard.secureSRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A17x1_A7x1.daughterboard.secure_region**
Allow TrustZone secure/normal bus signals to be routed separately.
Type: `TZSwitch`.

**FVP_VE_Cortex_A17x1_A7x1.daughterboard.sram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A17x1_A7x1.daughterboard.vedcc**
Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

**FVP_VE_Cortex_A17x1_A7x1.motherboard**
Model inspired by the ARM Versatile Express Motherboard.
Type: `VEMotherBoard`.

**FVP_VE_Cortex_A17x1_A7x1.motherboard.Timer_0_1**
ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_VE_Cortex_A17x1_A7x1.motherboard.Timer_0_1.clk_div0**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A17x1_A7x1.motherboard.Timer_0_1.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A17x1_A7x1.motherboard.Timer_0_1.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_VE_Cortex_A17x1_A7x1.motherboard.Timer_0_1.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_VE_Cortex_A17x1_A7x1.motherboard.Timer_2_3**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_VE_Cortex_A17x1_A7x1.motherboard.Timer_2_3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A17x1_A7x1.motherboard.Timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A17x1_A7x1.motherboard.Timer_2_3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_VE_Cortex_A17x1_A7x1.motherboard.Timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_VE_Cortex_A17x1_A7x1.motherboard.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**FVP_VE_Cortex_A17x1_A7x1.motherboard.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A17x1_A7x1.motherboard.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A17x1_A7x1.motherboard.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A17x1_A7x1.motherboard.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A17x1_A7x1.motherboard.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A17x1_A7x1.motherboard.dummy_local_dap_rom**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A17x1_A7x1.motherboard.dummy_ram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A17x1_A7x1.motherboard.dummy_usb**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A17x1_A7x1.motherboard.flash0**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_VE_Cortex_A17x1_A7x1.motherboard.flash1**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_VE_Cortex_A17x1_A7x1.motherboard.flashloader0**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A17x1_A7x1.motherboard.flashloader1**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_VE_Cortex_A17x1_A7x1.motherboard.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_VE_Cortex_A17x1_A7x1.motherboard.mmc**

Generic Multimedia Card.
Type: `MMC`.

**FVP_VE_Cortex_A17x1_A7x1.motherboard.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A17x1_A7x1.motherboard.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A17x1_A7x1.motherboard.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A17x1_A7x1.motherboard.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A17x1_A7x1.motherboard.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_A17x1_A7x1.motherboard.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A17x1_A7x1.motherboard.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A17x1_A7x1.motherboard.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A17x1_A7x1.motherboard.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A17x1_A7x1.motherboard.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A17x1_A7x1.motherboard.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_A17x1_A7x1.motherboard.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A17x1_A7x1.motherboard.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A17x1_A7x1.motherboard.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A17x1_A7x1.motherboard.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A17x1_A7x1.motherboard.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A17x1_A7x1.motherboard.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_A17x1_A7x1.motherboard.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A17x1_A7x1.motherboard.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A17x1_A7x1.motherboard.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A17x1_A7x1.motherboard.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A17x1_A7x1.motherboard.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A17x1_A7x1.motherboard.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_A17x1_A7x1.motherboard.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A17x1_A7x1.motherboard.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A17x1_A7x1.motherboard.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A17x1_A7x1.motherboard.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A17x1_A7x1.motherboard.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A17x1_A7x1.motherboard.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

**FVP_VE_Cortex_A17x1_A7x1.motherboard.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal()

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A17x1_A7x1.motherboard.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A17x1_A7x1.motherboard.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A17x1_A7x1.motherboard.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A17x1_A7x1.motherboard.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_VE_Cortex_A17x1_A7x1.motherboard.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A17x1_A7x1.motherboard.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A17x1_A7x1.motherboard.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A17x1_A7x1.motherboard.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A17x1_A7x1.motherboard.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_VE_Cortex_A17x1_A7x1.motherboard.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A17x1_A7x1.motherboard.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_VE_Cortex_A17x1_A7x1.motherboard.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A17x1_A7x1.motherboard.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_VE_Cortex_A17x1_A7x1.motherboard.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

**FVP_VE_Cortex_A17x1_A7x1.motherboard.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A17x1_A7x1.motherboard.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A17x1_A7x1.motherboard.pl111_clcd.pl11x_clcd.timer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A17x1_A7x1.motherboard.pl111_clcd.pl11x_clcd.timer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A17x1_A7x1.motherboard.pl180_mci**
ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

**FVP_VE_Cortex_A17x1_A7x1.motherboard.ps2keyboard**
Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

**FVP_VE_Cortex_A17x1_A7x1.motherboard.ps2keyboard.ps2_clocktimer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A17x1_A7x1.motherboard.ps2keyboard.ps2_clocktimer.timer**
A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A17x1_A7x1.motherboard.ps2keyboard.ps2_clocktimer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A17x1_A7x1.motherboard.ps2keyboard.ps2_clocktimer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A17x1_A7x1.motherboard.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

**FVP_VE_Cortex_A17x1_A7x1.motherboard.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A17x1_A7x1.motherboard.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A17x1_A7x1.motherboard.ps2mouse.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A17x1_A7x1.motherboard.ps2mouse.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A17x1_A7x1.motherboard.psram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A17x1_A7x1.motherboard.smsc_91c111**

SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_VE_Cortex_A17x1_A7x1.motherboard.sp805_wdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_VE_Cortex_A17x1_A7x1.motherboard.sp805_wdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A17x1_A7x1.motherboard.sp805_wdog.clocktimer.timer**
A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A17x1_A7x1.motherboard.sp805_wdog.clocktimer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A17x1_A7x1.motherboard.sp805_wdog.clocktimer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A17x1_A7x1.motherboard.sp810_sysctrl**
Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_VE_Cortex_A17x1_A7x1.motherboard.sp810_sysctrl.clkdiv_clk0**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A17x1_A7x1.motherboard.sp810_sysctrl.clkdiv_clk1**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A17x1_A7x1.motherboard.sp810_sysctrl.clkdiv_clk2**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A17x1_A7x1.motherboard.sp810_sysctrl.clkdiv_clk3**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**`FVP_VE_Cortex_A17x1_A7x1.motherboard.terminal_0`**
Telnet terminal interface.
Type: `TelnetTerminal`.

**`FVP_VE_Cortex_A17x1_A7x1.motherboard.terminal_1`**
Telnet terminal interface.
Type: `TelnetTerminal`.

**`FVP_VE_Cortex_A17x1_A7x1.motherboard.terminal_2`**
Telnet terminal interface.
Type: `TelnetTerminal`.

**`FVP_VE_Cortex_A17x1_A7x1.motherboard.terminal_3`**
Telnet terminal interface.
Type: `TelnetTerminal`.

**`FVP_VE_Cortex_A17x1_A7x1.motherboard.ve_sysregs`**
Type: `VE_SysRegs`.

**`FVP_VE_Cortex_A17x1_A7x1.motherboard.virtioblockdevice`**
virtio block device.
Type: `VirtioBlockDevice`.

**`FVP_VE_Cortex_A17x1_A7x1.motherboard.virtiop9device`**
virtio P9 server.
Type: `VirtioP9Device`.

**`FVP_VE_Cortex_A17x1_A7x1.motherboard.vis`**
Display window for VE using Visualisation library.
Type: `VEVisualisation`.

**`FVP_VE_Cortex_A17x1_A7x1.motherboard.vis.recorder`**
Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**`FVP_VE_Cortex_A17x1_A7x1.motherboard.vis.recorder.playbackDivider`**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**`FVP_VE_Cortex_A17x1_A7x1.motherboard.vis.recorder.playbackTimer`**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**`FVP_VE_Cortex_A17x1_A7x1.motherboard.vis.recorder.playbackTimer.timer`**
A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**`FVP_VE_Cortex_A17x1_A7x1.motherboard.vis.recorder.playbackTimer.timer.thread`**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**`FVP_VE_Cortex_A17x1_A7x1.motherboard.vis.recorder.playbackTimer.timer.thread_event`**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**`FVP_VE_Cortex_A17x1_A7x1.motherboard.vis.recorder.recordingDivider`**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**`FVP_VE_Cortex_A17x1_A7x1.motherboard.vram`**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

## 15.9 FVP_VE_Cortex-A17x2

List of instances in FVP_VE_Cortex-A17x2.

### FVP_VE_Cortex-A17x2 instances

**`FVP_VE_Cortex_A17x2`**
Top level component of the Cortex_A17x2 Versatile Express inspired model.
Type: `FVP_VE_Cortex_A17x2`.

**`FVP_VE_Cortex_A17x2.cluster`**
ARM Cortex-A17 Cluster CT model.
Type: `Cluster_ARM_Cortex-A17`.

**`FVP_VE_Cortex_A17x2.cluster.cpu0`**
ARM Cortex-A17 CT model.
Type: `ARM_Cortex-A17`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A17x2.cluster.cpu0.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_VE_Cortex_A17x2.cluster.cpu0.itlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_VE_Cortex_A17x2.cluster.cpu0.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A17x2.cluster.cpu0.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A17x2.cluster.cpu1**
ARM Cortex-A17 CT model.
Type: `ARM_Cortex-A17`.

**FVP_VE_Cortex_A17x2.cluster.cpu1.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_VE_Cortex_A17x2.cluster.cpu1.itlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_VE_Cortex_A17x2.cluster.cpu1.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A17x2.cluster.cpu1.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A17x2.cluster.l2_cache**
PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A17x2.daughterboard**
Daughtercard, inspired by the ARM Versatile Express development platform.
Type: `VEDaughterBoard`.

**FVP_VE_Cortex_A17x2.daughterboard.clockCLCD**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A17x2.daughterboard.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A17x2.daughterboard.dmc**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A17x2.daughterboard.dmc_phy**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A17x2.daughterboard.dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A17x2.daughterboard.dram_aliased**

Allow TrustZone secure/normal bus signals to be routed separately.
Type: `TZSwitch`.

**FVP_VE_Cortex_A17x2.daughterboard.dram_limit_4**

Allow TrustZone secure/normal bus signals to be routed separately.
Type: `TZSwitch`.

**FVP_VE_Cortex_A17x2.daughterboard.dram_limit_8**

Allow TrustZone secure/normal bus signals to be routed separately.
Type: `TZSwitch`.

**FVP_VE_Cortex_A17x2.daughterboard.exclusive_monitor**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_VE_Cortex_A17x2.daughterboard.hdlcd**

ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).
Type: `PL370_HDLCD`.

**FVP_VE_Cortex_A17x2.daughterboard.hdlcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A17x2.daughterboard.hdlcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A17x2.daughterboard.hdlcd.timer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A17x2.daughterboard.hdlcd.timer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A17x2.daughterboard.introuter**
Interrupt Mapping peripheral (non-cascaded).
Type: `VEInterruptMapper`.

**FVP_VE_Cortex_A17x2.daughterboard.nonsecure_region**
Allow TrustZone secure/normal bus signals to be routed separately.
Type: `TZSwitch`.

**FVP_VE_Cortex_A17x2.daughterboard.secureDRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A17x2.daughterboard.secureRO**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_VE_Cortex_A17x2.daughterboard.secureROloader**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_VE_Cortex_A17x2.daughterboard.secureSRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A17x2.daughterboard.secure_region**
Allow TrustZone secure/normal bus signals to be routed separately.
Type: `TZSwitch`.

**FVP_VE_Cortex_A17x2.daughterboard.sram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A17x2.daughterboard.vedcc**
Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

**FVP_VE_Cortex_A17x2.gic400**
GIC-400 Generic Interrupt Controller.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `GIC_400`.

**FVP_VE_Cortex_A17x2.globalcounter**

Memory Mapped Counter Module for Generic Timers.
Type: `MemoryMappedCounterModule`.

**FVP_VE_Cortex_A17x2.motherboard**

Model inspired by the ARM Versatile Express Motherboard.
Type: `VEMotherBoard`.

**FVP_VE_Cortex_A17x2.motherboard.Timer_0_1**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_VE_Cortex_A17x2.motherboard.Timer_0_1.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A17x2.motherboard.Timer_0_1.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A17x2.motherboard.Timer_0_1.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_VE_Cortex_A17x2.motherboard.Timer_0_1.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_VE_Cortex_A17x2.motherboard.Timer_2_3**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_VE_Cortex_A17x2.motherboard.Timer_2_3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A17x2.motherboard.Timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ClockDivider`.

**FVP_VE_Cortex_A17x2.motherboard.Timer_2_3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_VE_Cortex_A17x2.motherboard.Timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_VE_Cortex_A17x2.motherboard.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**FVP_VE_Cortex_A17x2.motherboard.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A17x2.motherboard.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A17x2.motherboard.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A17x2.motherboard.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A17x2.motherboard.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A17x2.motherboard.dummy_local_dap_rom**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A17x2.motherboard.dummy_ram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A17x2.motherboard.dummy_usb**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A17x2.motherboard.flash0**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_VE_Cortex_A17x2.motherboard.flash1**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_VE_Cortex_A17x2.motherboard.flashloader0**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_VE_Cortex_A17x2.motherboard.flashloader1**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_VE_Cortex_A17x2.motherboard.hostbridge**
Host Socket Interface Component.
Type: `HostBridge`.

**FVP_VE_Cortex_A17x2.motherboard.mmc**
Generic Multimedia Card.
Type: `MMC`.

**FVP_VE_Cortex_A17x2.motherboard.mmc.timer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A17x2.motherboard.mmc.timer.timer**
A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A17x2.motherboard.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A17x2.motherboard.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A17x2.motherboard.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_A17x2.motherboard.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A17x2.motherboard.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A17x2.motherboard.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A17x2.motherboard.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A17x2.motherboard.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A17x2.motherboard.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A17x2.motherboard.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A17x2.motherboard.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A17x2.motherboard.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A17x2.motherboard.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A17x2.motherboard.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A17x2.motherboard.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_A17x2.motherboard.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A17x2.motherboard.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A17x2.motherboard.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A17x2.motherboard.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A17x2.motherboard.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A17x2.motherboard.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_A17x2.motherboard.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A17x2.motherboard.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A17x2.motherboard.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A17x2.motherboard.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A17x2.motherboard.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A17x2.motherboard.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

**FVP_VE_Cortex_A17x2.motherboard.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A17x2.motherboard.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A17x2.motherboard.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A17x2.motherboard.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A17x2.motherboard.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_VE_Cortex_A17x2.motherboard.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A17x2.motherboard.pl041_aaci.timer.timer**
A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A17x2.motherboard.pl041_aaci.timer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A17x2.motherboard.pl041_aaci.timer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A17x2.motherboard.pl050_kmi0**
ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_VE_Cortex_A17x2.motherboard.pl050_kmi0.clk_divider**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A17x2.motherboard.pl050_kmi1**
ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_VE_Cortex_A17x2.motherboard.pl050_kmi1.clk_divider**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A17x2.motherboard.pl111_clcd**
ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_VE_Cortex_A17x2.motherboard.pl111_clcd.pl11x_clcd**
Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A17x2.motherboard.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A17x2.motherboard.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A17x2.motherboard.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A17x2.motherboard.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A17x2.motherboard.pl180_mci**

ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

**FVP_VE_Cortex_A17x2.motherboard.ps2keyboard**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

**FVP_VE_Cortex_A17x2.motherboard.ps2keyboard.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A17x2.motherboard.ps2keyboard.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A17x2.motherboard.ps2keyboard.ps2_clocktimer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A17x2.motherboard.ps2keyboard.ps2_clocktimer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A17x2.motherboard.ps2mouse**
Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

**FVP_VE_Cortex_A17x2.motherboard.ps2mouse.ps2_clocktimer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A17x2.motherboard.ps2mouse.ps2_clocktimer.timer**
A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A17x2.motherboard.ps2mouse.ps2_clocktimer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A17x2.motherboard.ps2mouse.ps2_clocktimer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A17x2.motherboard.psram**
RAM device, can be dynamic or static ram.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `RAMDevice`.

**FVP_VE_Cortex_A17x2.motherboard.smsc_91c111**

SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_VE_Cortex_A17x2.motherboard.sp805_wdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_VE_Cortex_A17x2.motherboard.sp805_wdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A17x2.motherboard.sp805_wdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A17x2.motherboard.sp805_wdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A17x2.motherboard.sp805_wdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A17x2.motherboard.sp810_sysctrl**

Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_VE_Cortex_A17x2.motherboard.sp810_sysctrl.clkdiv_clk0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A17x2.motherboard.sp810_sysctrl.clkdiv_clk1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A17x2.motherboard.sp810_sysctrl.clkdiv_clk2**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A17x2.motherboard.sp810_sysctrl.clkdiv_clk3**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A17x2.motherboard.terminal_0**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_A17x2.motherboard.terminal_1**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_A17x2.motherboard.terminal_2**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_A17x2.motherboard.terminal_3**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_A17x2.motherboard.ve_sysregs**

Type: `VE_SysRegs`.

**FVP_VE_Cortex_A17x2.motherboard.virtioblockdevice**

virtio block device.
Type: `VirtioBlockDevice`.

**FVP_VE_Cortex_A17x2.motherboard.virtiop9device**

virtio P9 server.
Type: `VirtioP9Device`.

**FVP_VE_Cortex_A17x2.motherboard.vis**

Display window for VE using Visualisation library.
Type: `VEVisualisation`.

**FVP_VE_Cortex_A17x2.motherboard.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A17x2.motherboard.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A17x2.motherboard.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A17x2.motherboard.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A17x2.motherboard.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A17x2.motherboard.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A17x2.motherboard.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A17x2.motherboard.vram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

# 15.10 FVP_VE_Cortex-A17x4

List of instances in FVP_VE_Cortex-A17x4.

## FVP_VE_Cortex-A17x4 instances

**FVP_VE_Cortex_A17x4**

Top level component of the Cortex_A17x4 Versatile Express inspired model.
Type: `FVP_VE_Cortex_A17x4`.

**FVP_VE_Cortex_A17x4.cluster**

ARM Cortex-A17 Cluster CT model.
Type: `Cluster_ARM_Cortex-A17`.

**FVP_VE_Cortex_A17x4.cluster.cpu0**

ARM Cortex-A17 CT model.
Type: `ARM_Cortex-A17`.

**FVP_VE_Cortex_A17x4.cluster.cpu0.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_VE_Cortex_A17x4.cluster.cpu0.itlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_VE_Cortex_A17x4.cluster.cpu0.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A17x4.cluster.cpu0.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A17x4.cluster.cpu1**

ARM Cortex-A17 CT model.
Type: `ARM_Cortex-A17`.

**FVP_VE_Cortex_A17x4.cluster.cpu1.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_VE_Cortex_A17x4.cluster.cpu1.itlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_VE_Cortex_A17x4.cluster.cpu1.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A17x4.cluster.cpu1.l1icache**

PV Cache.
Type: `PVCache`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A17x4.cluster.cpu2**
ARM Cortex-A17 CT model.
Type: ARM_Cortex-A17.

**FVP_VE_Cortex_A17x4.cluster.cpu2.dtlb**
TLB - instruction, data or unified.
Type: TlbCadi.

**FVP_VE_Cortex_A17x4.cluster.cpu2.itlb**
TLB - instruction, data or unified.
Type: TlbCadi.

**FVP_VE_Cortex_A17x4.cluster.cpu2.l1dcache**
PV Cache.
Type: PVCache.

**FVP_VE_Cortex_A17x4.cluster.cpu2.l1icache**
PV Cache.
Type: PVCache.

**FVP_VE_Cortex_A17x4.cluster.cpu3**
ARM Cortex-A17 CT model.
Type: ARM_Cortex-A17.

**FVP_VE_Cortex_A17x4.cluster.cpu3.dtlb**
TLB - instruction, data or unified.
Type: TlbCadi.

**FVP_VE_Cortex_A17x4.cluster.cpu3.itlb**
TLB - instruction, data or unified.
Type: TlbCadi.

**FVP_VE_Cortex_A17x4.cluster.cpu3.l1dcache**
PV Cache.
Type: PVCache.

**FVP_VE_Cortex_A17x4.cluster.cpu3.l1icache**
PV Cache.
Type: PVCache.

**FVP_VE_Cortex_A17x4.cluster.l2_cache**
PV Cache.
Type: PVCache.

**FVP_VE_Cortex_A17x4.daughterboard**
Daughtercard, inspired by the ARM Versatile Express development platform.
Type: VEDaughterBoard.

**FVP_VE_Cortex_A17x4.daughterboard.clockCLCD**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ClockDivider`.

**FVP_VE_Cortex_A17x4.daughterboard.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A17x4.daughterboard.dmc**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A17x4.daughterboard.dmc_phy**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A17x4.daughterboard.dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A17x4.daughterboard.dram_aliased**

Allow TrustZone secure/normal bus signals to be routed separately.
Type: `TZSwitch`.

**FVP_VE_Cortex_A17x4.daughterboard.dram_limit_4**

Allow TrustZone secure/normal bus signals to be routed separately.
Type: `TZSwitch`.

**FVP_VE_Cortex_A17x4.daughterboard.dram_limit_8**

Allow TrustZone secure/normal bus signals to be routed separately.
Type: `TZSwitch`.

**FVP_VE_Cortex_A17x4.daughterboard.exclusive_monitor**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_VE_Cortex_A17x4.daughterboard.hdlcd**

ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).
Type: `PL370_HDLCD`.

**FVP_VE_Cortex_A17x4.daughterboard.hdlcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A17x4.daughterboard.hdlcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A17x4.daughterboard.hdlcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A17x4.daughterboard.hdlcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A17x4.daughterboard.introuter**

Interrupt Mapping peripheral (non-cascaded).
Type: `VEInterruptMapper`.

**FVP_VE_Cortex_A17x4.daughterboard.nonsecure_region**

Allow TrustZone secure/normal bus signals to be routed separately.
Type: `TZSwitch`.

**FVP_VE_Cortex_A17x4.daughterboard.secureDRAM**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A17x4.daughterboard.secureRO**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_VE_Cortex_A17x4.daughterboard.secureROloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_VE_Cortex_A17x4.daughterboard.secureSRAM**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A17x4.daughterboard.secure_region**

Allow TrustZone secure/normal bus signals to be routed separately.
Type: `TZSwitch`.

**FVP_VE_Cortex_A17x4.daughterboard.sram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A17x4.daughterboard.vedcc**

Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A17x4.gic400**

GIC-400 Generic Interrupt Controller.
Type: `GIC_400`.

**FVP_VE_Cortex_A17x4.globalcounter**

Memory Mapped Counter Module for Generic Timers.
Type: `MemoryMappedCounterModule`.

**FVP_VE_Cortex_A17x4.motherboard**

Model inspired by the ARM Versatile Express Motherboard.
Type: `VEMotherBoard`.

**FVP_VE_Cortex_A17x4.motherboard.Timer_0_1**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_VE_Cortex_A17x4.motherboard.Timer_0_1.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A17x4.motherboard.Timer_0_1.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A17x4.motherboard.Timer_0_1.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_VE_Cortex_A17x4.motherboard.Timer_0_1.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_VE_Cortex_A17x4.motherboard.Timer_2_3**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_VE_Cortex_A17x4.motherboard.Timer_2_3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A17x4.motherboard.Timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**`FVP_VE_Cortex_A17x4.motherboard.Timer_2_3.counter0`**
Internal component used by SP804 Timer module.
Type: `CounterModule`.

**`FVP_VE_Cortex_A17x4.motherboard.Timer_2_3.counter1`**
Internal component used by SP804 Timer module.
Type: `CounterModule`.

**`FVP_VE_Cortex_A17x4.motherboard.audioout`**
SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**`FVP_VE_Cortex_A17x4.motherboard.clock100Hz`**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**`FVP_VE_Cortex_A17x4.motherboard.clock24MHz`**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**`FVP_VE_Cortex_A17x4.motherboard.clock35MHz`**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**`FVP_VE_Cortex_A17x4.motherboard.clock50Hz`**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**`FVP_VE_Cortex_A17x4.motherboard.clockCLCD`**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A17x4.motherboard.dummy_local_dap_rom**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A17x4.motherboard.dummy_ram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A17x4.motherboard.dummy_usb**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A17x4.motherboard.flash0**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_VE_Cortex_A17x4.motherboard.flash1**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_VE_Cortex_A17x4.motherboard.flashloader0**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_VE_Cortex_A17x4.motherboard.flashloader1**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_VE_Cortex_A17x4.motherboard.hostbridge**
Host Socket Interface Component.
Type: `HostBridge`.

**FVP_VE_Cortex_A17x4.motherboard.mmc**
Generic Multimedia Card.
Type: `MMC`.

**FVP_VE_Cortex_A17x4.motherboard.mmc.timer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A17x4.motherboard.mmc.timer.timer**
A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A17x4.motherboard.mmc.timer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A17x4.motherboard.mmc.timer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A17x4.motherboard.pl011_uart0**
ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_A17x4.motherboard.pl011_uart0.clk_divider**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A17x4.motherboard.pl011_uart0.timer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A17x4.motherboard.pl011_uart0.timer.timer**
A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A17x4.motherboard.pl011_uart0.timer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A17x4.motherboard.pl011_uart0.timer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A17x4.motherboard.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_A17x4.motherboard.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A17x4.motherboard.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A17x4.motherboard.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A17x4.motherboard.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A17x4.motherboard.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A17x4.motherboard.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_A17x4.motherboard.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A17x4.motherboard.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A17x4.motherboard.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A17x4.motherboard.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A17x4.motherboard.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A17x4.motherboard.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_A17x4.motherboard.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A17x4.motherboard.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A17x4.motherboard.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A17x4.motherboard.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A17x4.motherboard.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A17x4.motherboard.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

**FVP_VE_Cortex_A17x4.motherboard.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A17x4.motherboard.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A17x4.motherboard.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A17x4.motherboard.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A17x4.motherboard.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_VE_Cortex_A17x4.motherboard.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A17x4.motherboard.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A17x4.motherboard.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A17x4.motherboard.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A17x4.motherboard.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_VE_Cortex_A17x4.motherboard.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A17x4.motherboard.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_VE_Cortex_A17x4.motherboard.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A17x4.motherboard.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_VE_Cortex_A17x4.motherboard.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

**FVP_VE_Cortex_A17x4.motherboard.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A17x4.motherboard.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A17x4.motherboard.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A17x4.motherboard.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A17x4.motherboard.pl180_mci**

ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

**FVP_VE_Cortex_A17x4.motherboard.ps2keyboard**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A17x4.motherboard.ps2keyboard.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A17x4.motherboard.ps2keyboard.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A17x4.motherboard.ps2keyboard.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A17x4.motherboard.ps2keyboard.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A17x4.motherboard.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

**FVP_VE_Cortex_A17x4.motherboard.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A17x4.motherboard.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A17x4.motherboard.ps2mouse.ps2_clocktimer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A17x4.motherboard.ps2mouse.ps2_clocktimer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A17x4.motherboard.psram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A17x4.motherboard.smsc_91c111**
SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_VE_Cortex_A17x4.motherboard.sp805_wdog**
ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_VE_Cortex_A17x4.motherboard.sp805_wdog.clocktimer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A17x4.motherboard.sp805_wdog.clocktimer.timer**
A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A17x4.motherboard.sp805_wdog.clocktimer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A17x4.motherboard.sp805_wdog.clocktimer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A17x4.motherboard.sp810_sysctrl**

Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_VE_Cortex_A17x4.motherboard.sp810_sysctrl.clkdiv_clk0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A17x4.motherboard.sp810_sysctrl.clkdiv_clk1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A17x4.motherboard.sp810_sysctrl.clkdiv_clk2**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A17x4.motherboard.sp810_sysctrl.clkdiv_clk3**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A17x4.motherboard.terminal_0**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_A17x4.motherboard.terminal_1**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_A17x4.motherboard.terminal_2**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_A17x4.motherboard.terminal_3**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_A17x4.motherboard.ve_sysregs**

Type: `VE_SysRegs`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A17x4.motherboard.virtioblockdevice**

virtio block device.
Type: `VirtioBlockDevice`.

**FVP_VE_Cortex_A17x4.motherboard.virtiop9device**

virtio P9 server.
Type: `VirtioP9Device`.

**FVP_VE_Cortex_A17x4.motherboard.vis**

Display window for VE using Visualisation library.
Type: `VEVisualisation`.

**FVP_VE_Cortex_A17x4.motherboard.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_VE_Cortex_A17x4.motherboard.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A17x4.motherboard.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A17x4.motherboard.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A17x4.motherboard.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A17x4.motherboard.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A17x4.motherboard.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A17x4.motherboard.vram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

# 15.11 FVP_VE_Cortex-A17x4-A7x4

List of instances in FVP_VE_Cortex-A17x4-A7x4.

## FVP_VE_Cortex-A17x4-A7x4 instances

**FVP_VE_Cortex_A17x4_A7x4**

Top level component of the Cortex A17x4 A7x4 Versatile Express inspired model.
Type: `FVP_VE_Cortex_A17x4_A7x4`.

**FVP_VE_Cortex_A17x4_A7x4.coretile**

Dual cluster ARM Cortex-A17x4 and ARM Cortex-A7x4 Core Tile.
Type: `ARM_Cortex_A17x4_A7x4_CT`.

**FVP_VE_Cortex_A17x4_A7x4.coretile.cci400**

Cache Coherent Interconnect for AXI4 ACE.
Type: `CCI400`.

**FVP_VE_Cortex_A17x4_A7x4.coretile.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A17x4_A7x4.coretile.clockdivider1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A17x4_A7x4.coretile.cluster0**

ARM Cortex-A17 Cluster CT model.
Type: `Cluster_ARM_Cortex-A17`.

**FVP_VE_Cortex_A17x4_A7x4.coretile.cluster0.cpu0**

ARM Cortex-A17 CT model.
Type: `ARM_Cortex-A17`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A17x4_A7x4.coretile.cluster0.cpu0.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_VE_Cortex_A17x4_A7x4.coretile.cluster0.cpu0.itlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_VE_Cortex_A17x4_A7x4.coretile.cluster0.cpu0.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A17x4_A7x4.coretile.cluster0.cpu0.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A17x4_A7x4.coretile.cluster0.cpu1**
ARM Cortex-A17 CT model.
Type: `ARM_Cortex-A17`.

**FVP_VE_Cortex_A17x4_A7x4.coretile.cluster0.cpu1.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_VE_Cortex_A17x4_A7x4.coretile.cluster0.cpu1.itlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_VE_Cortex_A17x4_A7x4.coretile.cluster0.cpu1.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A17x4_A7x4.coretile.cluster0.cpu1.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A17x4_A7x4.coretile.cluster0.cpu2**
ARM Cortex-A17 CT model.
Type: `ARM_Cortex-A17`.

**FVP_VE_Cortex_A17x4_A7x4.coretile.cluster0.cpu2.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_VE_Cortex_A17x4_A7x4.coretile.cluster0.cpu2.itlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_VE_Cortex_A17x4_A7x4.coretile.cluster0.cpu2.l1dcache**
PV Cache.
Type: `PVCache`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A17x4_A7x4.coretile.cluster0.cpu2.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A17x4_A7x4.coretile.cluster0.cpu3**
ARM Cortex-A17 CT model.
Type: `ARM_Cortex-A17`.

**FVP_VE_Cortex_A17x4_A7x4.coretile.cluster0.cpu3.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_VE_Cortex_A17x4_A7x4.coretile.cluster0.cpu3.itlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_VE_Cortex_A17x4_A7x4.coretile.cluster0.cpu3.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A17x4_A7x4.coretile.cluster0.cpu3.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A17x4_A7x4.coretile.cluster0.l2_cache**
PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A17x4_A7x4.coretile.cluster1**
ARM Cortex-A7 Cluster CT model.
Type: `Cluster_ARM_Cortex-A7`.

**FVP_VE_Cortex_A17x4_A7x4.coretile.cluster1.cpu0**
ARM Cortex-A7 CT model.
Type: `ARM_Cortex-A7`.

**FVP_VE_Cortex_A17x4_A7x4.coretile.cluster1.cpu0.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_VE_Cortex_A17x4_A7x4.coretile.cluster1.cpu0.itlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_VE_Cortex_A17x4_A7x4.coretile.cluster1.cpu0.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A17x4_A7x4.coretile.cluster1.cpu0.l1icache**
PV Cache.
Type: `PVCache`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A17x4_A7x4.coretile.cluster1.cpu1**
ARM Cortex-A7 CT model.
Type: `ARM_Cortex-A7`.

**FVP_VE_Cortex_A17x4_A7x4.coretile.cluster1.cpu1.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_VE_Cortex_A17x4_A7x4.coretile.cluster1.cpu1.itlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_VE_Cortex_A17x4_A7x4.coretile.cluster1.cpu1.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A17x4_A7x4.coretile.cluster1.cpu1.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A17x4_A7x4.coretile.cluster1.cpu2**
ARM Cortex-A7 CT model.
Type: `ARM_Cortex-A7`.

**FVP_VE_Cortex_A17x4_A7x4.coretile.cluster1.cpu2.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_VE_Cortex_A17x4_A7x4.coretile.cluster1.cpu2.itlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_VE_Cortex_A17x4_A7x4.coretile.cluster1.cpu2.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A17x4_A7x4.coretile.cluster1.cpu2.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A17x4_A7x4.coretile.cluster1.cpu3**
ARM Cortex-A7 CT model.
Type: `ARM_Cortex-A7`.

**FVP_VE_Cortex_A17x4_A7x4.coretile.cluster1.cpu3.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_VE_Cortex_A17x4_A7x4.coretile.cluster1.cpu3.itlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A17x4_A7x4.coretile.cluster1.cpu3.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A17x4_A7x4.coretile.cluster1.cpu3.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A17x4_A7x4.coretile.cluster1.l2_cache**
PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A17x4_A7x4.coretile.dualclustersystemconfigurationblock**
Dual Cluster System Configuration Block.
Type: `DualClusterSystemConfigurationBlock`.

**FVP_VE_Cortex_A17x4_A7x4.coretile.globalcounter**
Memory Mapped Counter Module for Generic Timers.
Type: `MemoryMappedCounterModule`.

**FVP_VE_Cortex_A17x4_A7x4.coretile.v7_vgic**
System VGIC architecture version v7.
Type: `v7_VGIC`.

**FVP_VE_Cortex_A17x4_A7x4.daughterboard**
Daughtercard, inspired by the ARM Versatile Express development platform.
Type: `VEDaughterBoard`.

**FVP_VE_Cortex_A17x4_A7x4.daughterboard.clockCLCD**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A17x4_A7x4.daughterboard.clockdivider**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A17x4_A7x4.daughterboard.dmc**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A17x4_A7x4.daughterboard.dmc_phy**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A17x4_A7x4.daughterboard.dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A17x4_A7x4.daughterboard.dram_aliased**

Allow TrustZone secure/normal bus signals to be routed separately.
Type: `TZSwitch`.

**FVP_VE_Cortex_A17x4_A7x4.daughterboard.dram_limit_4**

Allow TrustZone secure/normal bus signals to be routed separately.
Type: `TZSwitch`.

**FVP_VE_Cortex_A17x4_A7x4.daughterboard.dram_limit_8**

Allow TrustZone secure/normal bus signals to be routed separately.
Type: `TZSwitch`.

**FVP_VE_Cortex_A17x4_A7x4.daughterboard.exclusive_monitor**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_VE_Cortex_A17x4_A7x4.daughterboard.hdlcd**

ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).
Type: `PL370_HDLCD`.

**FVP_VE_Cortex_A17x4_A7x4.daughterboard.hdlcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A17x4_A7x4.daughterboard.hdlcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A17x4_A7x4.daughterboard.hdlcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A17x4_A7x4.daughterboard.hdlcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A17x4_A7x4.daughterboard.introuter**

Interrupt Mapping peripheral (non-cascaded).
Type: `VEInterruptMapper`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A17x4_A7x4.daughterboard.nonsecure_region**
Allow TrustZone secure/normal bus signals to be routed separately.
Type: `TZSwitch`.

**FVP_VE_Cortex_A17x4_A7x4.daughterboard.secureDRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A17x4_A7x4.daughterboard.secureRO**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_VE_Cortex_A17x4_A7x4.daughterboard.secureROloader**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_VE_Cortex_A17x4_A7x4.daughterboard.secureSRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A17x4_A7x4.daughterboard.secure_region**
Allow TrustZone secure/normal bus signals to be routed separately.
Type: `TZSwitch`.

**FVP_VE_Cortex_A17x4_A7x4.daughterboard.sram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A17x4_A7x4.daughterboard.vedcc**
Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard**
Model inspired by the ARM Versatile Express Motherboard.
Type: `VEMotherBoard`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.Timer_0_1**
ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.Timer_0_1.clk_div0**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.Timer_0_1.clk_div1**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A17x4_A7x4.motherboard.Timer_0_1.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.Timer_0_1.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.Timer_2_3**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.Timer_2_3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.Timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.Timer_2_3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.Timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A17x4_A7x4.motherboard.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.dummy_local_dap_rom**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.dummy_ram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.dummy_usb**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.flash0**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.flash1**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.flashloader0**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.flashloader1**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A17x4_A7x4.motherboard.mmc**

Generic Multimedia Card.
Type: `MMC`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ClockTimerThread`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A17x4_A7x4.motherboard.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A17x4_A7x4.motherboard.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A17x4_A7x4.motherboard.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A17x4_A7x4.motherboard.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `SchedulerThread`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.pl180_mci**

ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.ps2keyboard**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.ps2keyboard.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.ps2keyboard.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.ps2keyboard.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.ps2keyboard.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A17x4_A7x4.motherboard.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.ps2mouse.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.ps2mouse.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.psram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.smsc_91c111**

SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.sp805_wdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.sp805_wdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A17x4_A7x4.motherboard.sp805_wdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.sp805_wdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.sp805_wdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.sp810_sysctrl**

Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.sp810_sysctrl.clkdiv_clk0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.sp810_sysctrl.clkdiv_clk1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.sp810_sysctrl.clkdiv_clk2**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.sp810_sysctrl.clkdiv_clk3**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A17x4_A7x4.motherboard.terminal_0**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.terminal_1**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.terminal_2**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.terminal_3**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.ve_sysregs**
Type: `VE_SysRegs`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.virtioblockdevice**
virtio block device.
Type: `VirtioBlockDevice`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.virtiop9device**
virtio P9 server.
Type: `VirtioP9Device`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.vis**
Display window for VE using Visualisation library.
Type: `VEVisualisation`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.vis.recorder**
Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.vis.recorder.playbackDivider**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.vis.recorder.playbackTimer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A17x4_A7x4.motherboard.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A17x4_A7x4.motherboard.vram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

# 15.12 FVP_VE_Cortex-A5x1

List of instances in FVP_VE_Cortex-A5x1.

## FVP_VE_Cortex-A5x1 instances

**FVP_VE_Cortex_A5x1**

Top level component of the Cortex-A5x1 Versatile Express inspired model.
Type: `FVP_VE_Cortex_A5x1`.

**FVP_VE_Cortex_A5x1.cluster**

ARM CORTEXA5MP Cluster CT model.
Type: `Cluster_ARM_Cortex-A5MP`.

**FVP_VE_Cortex_A5x1.cluster.cpu0**

ARM CORTEXA5MP CT model.
Type: `ARM_Cortex-A5MP`.

**FVP_VE_Cortex_A5x1.cluster.cpu0.l1dcache**

PV Cache.
Type: `PVCache`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A5x1.cluster.cpu0.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A5x1.cluster.cpu0.utlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_VE_Cortex_A5x1.daughterboard**

Daughtercard, inspired by the ARM Versatile Express development platform.
Type: `VEDaughterBoard`.

**FVP_VE_Cortex_A5x1.daughterboard.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A5x1.daughterboard.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A5x1.daughterboard.dmc**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A5x1.daughterboard.dmc_phy**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A5x1.daughterboard.dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A5x1.daughterboard.dram_aliased**

Allow TrustZone secure/normal bus signals to be routed separately.
Type: `TZSwitch`.

**FVP_VE_Cortex_A5x1.daughterboard.dram_limit_4**

Allow TrustZone secure/normal bus signals to be routed separately.
Type: `TZSwitch`.

**FVP_VE_Cortex_A5x1.daughterboard.dram_limit_8**

Allow TrustZone secure/normal bus signals to be routed separately.
Type: `TZSwitch`.

**FVP_VE_Cortex_A5x1.daughterboard.exclusive_monitor**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A5x1.daughterboard.hdlcd**

ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).
Type: `PL370_HDLCD`.

**FVP_VE_Cortex_A5x1.daughterboard.hdlcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A5x1.daughterboard.hdlcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A5x1.daughterboard.hdlcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A5x1.daughterboard.hdlcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A5x1.daughterboard.introuter**

Interrupt Mapping peripheral (non-cascaded).
Type: `VEInterruptMapper`.

**FVP_VE_Cortex_A5x1.daughterboard.nonsecure_region**

Allow TrustZone secure/normal bus signals to be routed separately.
Type: `TZSwitch`.

**FVP_VE_Cortex_A5x1.daughterboard.secureDRAM**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A5x1.daughterboard.secureRO**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_VE_Cortex_A5x1.daughterboard.secureROloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A5x1.daughterboard.secureSRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A5x1.daughterboard.secure_region**
Allow TrustZone secure/normal bus signals to be routed separately.
Type: `TZSwitch`.

**FVP_VE_Cortex_A5x1.daughterboard.sram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A5x1.daughterboard.vedcc**
Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

**FVP_VE_Cortex_A5x1.motherboard**
Model inspired by the ARM Versatile Express Motherboard.
Type: `VEMotherBoard`.

**FVP_VE_Cortex_A5x1.motherboard.Timer_0_1**
ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_VE_Cortex_A5x1.motherboard.Timer_0_1.clk_div0**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A5x1.motherboard.Timer_0_1.clk_div1**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A5x1.motherboard.Timer_0_1.counter0**
Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_VE_Cortex_A5x1.motherboard.Timer_0_1.counter1**
Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_VE_Cortex_A5x1.motherboard.Timer_2_3**
ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_VE_Cortex_A5x1.motherboard.Timer_2_3.clk_div0**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A5x1.motherboard.Timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A5x1.motherboard.Timer_2_3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_VE_Cortex_A5x1.motherboard.Timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_VE_Cortex_A5x1.motherboard.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**FVP_VE_Cortex_A5x1.motherboard.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A5x1.motherboard.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A5x1.motherboard.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A5x1.motherboard.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A5x1.motherboard.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A5x1.motherboard.dummy_local_dap_rom**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A5x1.motherboard.dummy_ram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A5x1.motherboard.dummy_usb**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A5x1.motherboard.flash0**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_VE_Cortex_A5x1.motherboard.flash1**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_VE_Cortex_A5x1.motherboard.flashloader0**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_VE_Cortex_A5x1.motherboard.flashloader1**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_VE_Cortex_A5x1.motherboard.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_VE_Cortex_A5x1.motherboard.mmc**

Generic Multimedia Card.
Type: `MMC`.

**FVP_VE_Cortex_A5x1.motherboard.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A5x1.motherboard.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A5x1.motherboard.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A5x1.motherboard.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A5x1.motherboard.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_A5x1.motherboard.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A5x1.motherboard.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A5x1.motherboard.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A5x1.motherboard.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A5x1.motherboard.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A5x1.motherboard.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_A5x1.motherboard.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A5x1.motherboard.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A5x1.motherboard.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A5x1.motherboard.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A5x1.motherboard.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A5x1.motherboard.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_A5x1.motherboard.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A5x1.motherboard.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A5x1.motherboard.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A5x1.motherboard.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A5x1.motherboard.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A5x1.motherboard.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_A5x1.motherboard.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A5x1.motherboard.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ClockTimerThread`.

**FVP_VE_Cortex_A5x1.motherboard.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A5x1.motherboard.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A5x1.motherboard.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A5x1.motherboard.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

**FVP_VE_Cortex_A5x1.motherboard.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A5x1.motherboard.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A5x1.motherboard.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A5x1.motherboard.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A5x1.motherboard.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_VE_Cortex_A5x1.motherboard.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A5x1.motherboard.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A5x1.motherboard.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A5x1.motherboard.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A5x1.motherboard.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_VE_Cortex_A5x1.motherboard.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A5x1.motherboard.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_VE_Cortex_A5x1.motherboard.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A5x1.motherboard.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_VE_Cortex_A5x1.motherboard.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

**FVP_VE_Cortex_A5x1.motherboard.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A5x1.motherboard.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A5x1.motherboard.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A5x1.motherboard.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A5x1.motherboard.pl180_mci**

ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

**FVP_VE_Cortex_A5x1.motherboard.ps2keyboard**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A5x1.motherboard.ps2keyboard.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A5x1.motherboard.ps2keyboard.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A5x1.motherboard.ps2keyboard.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A5x1.motherboard.ps2keyboard.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A5x1.motherboard.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

**FVP_VE_Cortex_A5x1.motherboard.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A5x1.motherboard.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A5x1.motherboard.ps2mouse.ps2_clocktimer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A5x1.motherboard.ps2mouse.ps2_clocktimer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A5x1.motherboard.psram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A5x1.motherboard.smsc_91c111**
SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_VE_Cortex_A5x1.motherboard.sp805_wdog**
ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_VE_Cortex_A5x1.motherboard.sp805_wdog.clocktimer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A5x1.motherboard.sp805_wdog.clocktimer.timer**
A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A5x1.motherboard.sp805_wdog.clocktimer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A5x1.motherboard.sp805_wdog.clocktimer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A5x1.motherboard.sp810_sysctrl**

Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_VE_Cortex_A5x1.motherboard.sp810_sysctrl.clkdiv_clk0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A5x1.motherboard.sp810_sysctrl.clkdiv_clk1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A5x1.motherboard.sp810_sysctrl.clkdiv_clk2**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A5x1.motherboard.sp810_sysctrl.clkdiv_clk3**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A5x1.motherboard.terminal_0**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_A5x1.motherboard.terminal_1**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_A5x1.motherboard.terminal_2**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_A5x1.motherboard.terminal_3**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_A5x1.motherboard.ve_sysregs**

Type: `VE_SysRegs`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A5x1.motherboard.virtioblockdevice**

virtio block device.
Type: `VirtioBlockDevice`.

**FVP_VE_Cortex_A5x1.motherboard.virtiop9device**

virtio P9 server.
Type: `VirtioP9Device`.

**FVP_VE_Cortex_A5x1.motherboard.vis**

Display window for VE using Visualisation library.
Type: `VEVisualisation`.

**FVP_VE_Cortex_A5x1.motherboard.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_VE_Cortex_A5x1.motherboard.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A5x1.motherboard.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A5x1.motherboard.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A5x1.motherboard.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A5x1.motherboard.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A5x1.motherboard.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A5x1.motherboard.vram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A5x1.periph_clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

## 15.13 FVP_VE_Cortex-A5x2

List of instances in FVP_VE_Cortex-A5x2.

### FVP_VE_Cortex-A5x2 instances

**FVP_VE_Cortex_A5x2**

Top level component of the Cortex-A5x2 Versatile Express inspired model.
Type: `FVP_VE_Cortex_A5x2`.

**FVP_VE_Cortex_A5x2.cluster**

ARM CORTEXA5MP Cluster CT model.
Type: `Cluster_ARM_Cortex-A5MP`.

**FVP_VE_Cortex_A5x2.cluster.cpu0**

ARM CORTEXA5MP CT model.
Type: `ARM_Cortex-A5MP`.

**FVP_VE_Cortex_A5x2.cluster.cpu0.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A5x2.cluster.cpu0.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A5x2.cluster.cpu0.utlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_VE_Cortex_A5x2.cluster.cpu1**

ARM CORTEXA5MP CT model.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ARM_Cortex-A5MP`.

**FVP_VE_Cortex_A5x2.cluster.cpu1.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A5x2.cluster.cpu1.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A5x2.cluster.cpu1.utlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_VE_Cortex_A5x2.daughterboard**

Daughtercard, inspired by the ARM Versatile Express development platform.
Type: `VEDaughterBoard`.

**FVP_VE_Cortex_A5x2.daughterboard.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A5x2.daughterboard.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A5x2.daughterboard.dmc**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A5x2.daughterboard.dmc_phy**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A5x2.daughterboard.dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A5x2.daughterboard.dram_aliased**

Allow TrustZone secure/normal bus signals to be routed separately.
Type: `TZSwitch`.

**FVP_VE_Cortex_A5x2.daughterboard.dram_limit_4**

Allow TrustZone secure/normal bus signals to be routed separately.
Type: `TZSwitch`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A5x2.daughterboard.dram_limit_8**

Allow TrustZone secure/normal bus signals to be routed separately.
Type: `TZSwitch`.

**FVP_VE_Cortex_A5x2.daughterboard.exclusive_monitor**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_VE_Cortex_A5x2.daughterboard.hdlcd**

ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).
Type: `PL370_HDLCD`.

**FVP_VE_Cortex_A5x2.daughterboard.hdlcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A5x2.daughterboard.hdlcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A5x2.daughterboard.hdlcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A5x2.daughterboard.hdlcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A5x2.daughterboard.introuter**

Interrupt Mapping peripheral (non-cascaded).
Type: `VEInterruptMapper`.

**FVP_VE_Cortex_A5x2.daughterboard.nonsecure_region**

Allow TrustZone secure/normal bus signals to be routed separately.
Type: `TZSwitch`.

**FVP_VE_Cortex_A5x2.daughterboard.secureDRAM**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A5x2.daughterboard.secureRO**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_VE_Cortex_A5x2.daughterboard.secureROloader**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_VE_Cortex_A5x2.daughterboard.secureSRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A5x2.daughterboard.secure_region**
Allow TrustZone secure/normal bus signals to be routed separately.
Type: `TZSwitch`.

**FVP_VE_Cortex_A5x2.daughterboard.sram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A5x2.daughterboard.vedcc**
Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

**FVP_VE_Cortex_A5x2.motherboard**
Model inspired by the ARM Versatile Express Motherboard.
Type: `VEMotherBoard`.

**FVP_VE_Cortex_A5x2.motherboard.Timer_0_1**
ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_VE_Cortex_A5x2.motherboard.Timer_0_1.clk_div0**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A5x2.motherboard.Timer_0_1.clk_div1**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A5x2.motherboard.Timer_0_1.counter0**
Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_VE_Cortex_A5x2.motherboard.Timer_0_1.counter1**
Internal component used by SP804 Timer module.
Type: `CounterModule`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A5x2.motherboard.Timer_2_3**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_VE_Cortex_A5x2.motherboard.Timer_2_3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A5x2.motherboard.Timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A5x2.motherboard.Timer_2_3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_VE_Cortex_A5x2.motherboard.Timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_VE_Cortex_A5x2.motherboard.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**FVP_VE_Cortex_A5x2.motherboard.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A5x2.motherboard.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A5x2.motherboard.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A5x2.motherboard.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A5x2.motherboard.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A5x2.motherboard.dummy_local_dap_rom**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A5x2.motherboard.dummy_ram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A5x2.motherboard.dummy_usb**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A5x2.motherboard.flash0**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_VE_Cortex_A5x2.motherboard.flash1**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_VE_Cortex_A5x2.motherboard.flashloader0**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_VE_Cortex_A5x2.motherboard.flashloader1**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_VE_Cortex_A5x2.motherboard.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_VE_Cortex_A5x2.motherboard.mmc**

Generic Multimedia Card.
Type: `MMC`.

**FVP_VE_Cortex_A5x2.motherboard.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal()

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A5x2.motherboard.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A5x2.motherboard.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A5x2.motherboard.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A5x2.motherboard.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_A5x2.motherboard.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A5x2.motherboard.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A5x2.motherboard.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A5x2.motherboard.pl011_uart0.timer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A5x2.motherboard.pl011_uart0.timer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A5x2.motherboard.pl011_uart1**
ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_A5x2.motherboard.pl011_uart1.clk_divider**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A5x2.motherboard.pl011_uart1.timer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A5x2.motherboard.pl011_uart1.timer.timer**
A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A5x2.motherboard.pl011_uart1.timer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A5x2.motherboard.pl011_uart1.timer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A5x2.motherboard.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_A5x2.motherboard.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A5x2.motherboard.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A5x2.motherboard.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A5x2.motherboard.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A5x2.motherboard.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A5x2.motherboard.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_A5x2.motherboard.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A5x2.motherboard.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A5x2.motherboard.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A5x2.motherboard.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A5x2.motherboard.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A5x2.motherboard.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

**FVP_VE_Cortex_A5x2.motherboard.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A5x2.motherboard.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A5x2.motherboard.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A5x2.motherboard.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A5x2.motherboard.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_VE_Cortex_A5x2.motherboard.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A5x2.motherboard.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A5x2.motherboard.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A5x2.motherboard.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A5x2.motherboard.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_VE_Cortex_A5x2.motherboard.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A5x2.motherboard.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_VE_Cortex_A5x2.motherboard.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A5x2.motherboard.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_VE_Cortex_A5x2.motherboard.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

**FVP_VE_Cortex_A5x2.motherboard.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A5x2.motherboard.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A5x2.motherboard.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A5x2.motherboard.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A5x2.motherboard.pl180_mci**

ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A5x2.motherboard.ps2keyboard**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

**FVP_VE_Cortex_A5x2.motherboard.ps2keyboard.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A5x2.motherboard.ps2keyboard.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A5x2.motherboard.ps2keyboard.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A5x2.motherboard.ps2keyboard.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A5x2.motherboard.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

**FVP_VE_Cortex_A5x2.motherboard.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A5x2.motherboard.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A5x2.motherboard.ps2mouse.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A5x2.motherboard.ps2mouse.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A5x2.motherboard.psram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A5x2.motherboard.smsc_91c111**

SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_VE_Cortex_A5x2.motherboard.sp805_wdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_VE_Cortex_A5x2.motherboard.sp805_wdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A5x2.motherboard.sp805_wdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A5x2.motherboard.sp805_wdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `SchedulerThread`.

**FVP_VE_Cortex_A5x2.motherboard.sp805_wdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A5x2.motherboard.sp810_sysctrl**

Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_VE_Cortex_A5x2.motherboard.sp810_sysctrl.clkdiv_clk0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A5x2.motherboard.sp810_sysctrl.clkdiv_clk1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A5x2.motherboard.sp810_sysctrl.clkdiv_clk2**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A5x2.motherboard.sp810_sysctrl.clkdiv_clk3**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A5x2.motherboard.terminal_0**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_A5x2.motherboard.terminal_1**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_A5x2.motherboard.terminal_2**

Telnet terminal interface.
Type: `TelnetTerminal`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A5x2.motherboard.terminal_3**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_A5x2.motherboard.ve_sysregs**

Type: `VE_SysRegs`.

**FVP_VE_Cortex_A5x2.motherboard.virtioblockdevice**

virtio block device.
Type: `VirtioBlockDevice`.

**FVP_VE_Cortex_A5x2.motherboard.virtiop9device**

virtio P9 server.
Type: `VirtioP9Device`.

**FVP_VE_Cortex_A5x2.motherboard.vis**

Display window for VE using Visualisation library.
Type: `VEVisualisation`.

**FVP_VE_Cortex_A5x2.motherboard.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_VE_Cortex_A5x2.motherboard.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A5x2.motherboard.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A5x2.motherboard.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A5x2.motherboard.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `SchedulerThread`.

**`FVP_VE_Cortex_A5x2.motherboard.vis.recorder.playbackTimer.timer.thread_event`**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**`FVP_VE_Cortex_A5x2.motherboard.vis.recorder.recordingDivider`**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**`FVP_VE_Cortex_A5x2.motherboard.vram`**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**`FVP_VE_Cortex_A5x2.periph_clockdivider`**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

## 15.14 FVP_VE_Cortex-A5x4

List of instances in FVP_VE_Cortex-A5x4.

### FVP_VE_Cortex-A5x4 instances

**`FVP_VE_Cortex_A5x4`**

Top level component of the Cortex-A5x4 Versatile Express inspired model.
Type: `FVP_VE_Cortex_A5x4`.

**`FVP_VE_Cortex_A5x4.cluster`**

ARM CORTEXA5MP Cluster CT model.
Type: `Cluster_ARM_Cortex-A5MP`.

**`FVP_VE_Cortex_A5x4.cluster.cpu0`**

ARM CORTEXA5MP CT model.
Type: `ARM_Cortex-A5MP`.

**`FVP_VE_Cortex_A5x4.cluster.cpu0.l1dcache`**

PV Cache.
Type: `PVCache`.

**`FVP_VE_Cortex_A5x4.cluster.cpu0.l1icache`**

PV Cache.
Type: `PVCache`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A5x4.cluster.cpu0.utlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_VE_Cortex_A5x4.cluster.cpu1**
ARM CORTEXA5MP CT model.
Type: `ARM_Cortex-A5MP`.

**FVP_VE_Cortex_A5x4.cluster.cpu1.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A5x4.cluster.cpu1.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A5x4.cluster.cpu1.utlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_VE_Cortex_A5x4.cluster.cpu2**
ARM CORTEXA5MP CT model.
Type: `ARM_Cortex-A5MP`.

**FVP_VE_Cortex_A5x4.cluster.cpu2.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A5x4.cluster.cpu2.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A5x4.cluster.cpu2.utlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_VE_Cortex_A5x4.cluster.cpu3**
ARM CORTEXA5MP CT model.
Type: `ARM_Cortex-A5MP`.

**FVP_VE_Cortex_A5x4.cluster.cpu3.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A5x4.cluster.cpu3.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A5x4.cluster.cpu3.utlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A5x4.daughterboard**

Daughtercard, inspired by the ARM Versatile Express development platform.
Type: `VEDaughterBoard`.

**FVP_VE_Cortex_A5x4.daughterboard.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A5x4.daughterboard.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A5x4.daughterboard.dmc**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A5x4.daughterboard.dmc_phy**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A5x4.daughterboard.dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A5x4.daughterboard.dram_aliased**

Allow TrustZone secure/normal bus signals to be routed separately.
Type: `TZSwitch`.

**FVP_VE_Cortex_A5x4.daughterboard.dram_limit_4**

Allow TrustZone secure/normal bus signals to be routed separately.
Type: `TZSwitch`.

**FVP_VE_Cortex_A5x4.daughterboard.dram_limit_8**

Allow TrustZone secure/normal bus signals to be routed separately.
Type: `TZSwitch`.

**FVP_VE_Cortex_A5x4.daughterboard.exclusive_monitor**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_VE_Cortex_A5x4.daughterboard.hdlcd**

ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).
Type: `PL370_HDLCD`.

**FVP_VE_Cortex_A5x4.daughterboard.hdlcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal()

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A5x4.daughterboard.hdlcd.timer.timer**
A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A5x4.daughterboard.hdlcd.timer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A5x4.daughterboard.hdlcd.timer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A5x4.daughterboard.introuter**
Interrupt Mapping peripheral (non-cascaded).
Type: `VEInterruptMapper`.

**FVP_VE_Cortex_A5x4.daughterboard.nonsecure_region**
Allow TrustZone secure/normal bus signals to be routed separately.
Type: `TZSwitch`.

**FVP_VE_Cortex_A5x4.daughterboard.secureDRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A5x4.daughterboard.secureRO**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_VE_Cortex_A5x4.daughterboard.secureROloader**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_VE_Cortex_A5x4.daughterboard.secureSRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A5x4.daughterboard.secure_region**
Allow TrustZone secure/normal bus signals to be routed separately.
Type: `TZSwitch`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A5x4.daughterboard.sram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A5x4.daughterboard.vedcc**
Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

**FVP_VE_Cortex_A5x4.motherboard**
Model inspired by the ARM Versatile Express Motherboard.
Type: `VEMotherBoard`.

**FVP_VE_Cortex_A5x4.motherboard.Timer_0_1**
ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_VE_Cortex_A5x4.motherboard.Timer_0_1.clk_div0**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A5x4.motherboard.Timer_0_1.clk_div1**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A5x4.motherboard.Timer_0_1.counter0**
Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_VE_Cortex_A5x4.motherboard.Timer_0_1.counter1**
Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_VE_Cortex_A5x4.motherboard.Timer_2_3**
ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_VE_Cortex_A5x4.motherboard.Timer_2_3.clk_div0**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A5x4.motherboard.Timer_2_3.clk_div1**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**`FVP_VE_Cortex_A5x4.motherboard.Timer_2_3.counter0`**
Internal component used by SP804 Timer module.
Type: `CounterModule`.

**`FVP_VE_Cortex_A5x4.motherboard.Timer_2_3.counter1`**
Internal component used by SP804 Timer module.
Type: `CounterModule`.

**`FVP_VE_Cortex_A5x4.motherboard.audioout`**
SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**`FVP_VE_Cortex_A5x4.motherboard.clock100Hz`**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**`FVP_VE_Cortex_A5x4.motherboard.clock24MHz`**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**`FVP_VE_Cortex_A5x4.motherboard.clock35MHz`**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**`FVP_VE_Cortex_A5x4.motherboard.clock50Hz`**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**`FVP_VE_Cortex_A5x4.motherboard.clockCLCD`**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A5x4.motherboard.dummy_local_dap_rom**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A5x4.motherboard.dummy_ram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A5x4.motherboard.dummy_usb**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A5x4.motherboard.flash0**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_VE_Cortex_A5x4.motherboard.flash1**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_VE_Cortex_A5x4.motherboard.flashloader0**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_VE_Cortex_A5x4.motherboard.flashloader1**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_VE_Cortex_A5x4.motherboard.hostbridge**
Host Socket Interface Component.
Type: `HostBridge`.

**FVP_VE_Cortex_A5x4.motherboard.mmc**
Generic Multimedia Card.
Type: `MMC`.

**FVP_VE_Cortex_A5x4.motherboard.mmc.timer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A5x4.motherboard.mmc.timer.timer**
A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A5x4.motherboard.mmc.timer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A5x4.motherboard.mmc.timer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A5x4.motherboard.pl011_uart0**
ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_A5x4.motherboard.pl011_uart0.clk_divider**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A5x4.motherboard.pl011_uart0.timer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A5x4.motherboard.pl011_uart0.timer.timer**
A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A5x4.motherboard.pl011_uart0.timer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A5x4.motherboard.pl011_uart0.timer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A5x4.motherboard.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_A5x4.motherboard.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A5x4.motherboard.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A5x4.motherboard.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A5x4.motherboard.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A5x4.motherboard.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A5x4.motherboard.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_A5x4.motherboard.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A5x4.motherboard.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A5x4.motherboard.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A5x4.motherboard.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A5x4.motherboard.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A5x4.motherboard.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_A5x4.motherboard.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A5x4.motherboard.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A5x4.motherboard.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A5x4.motherboard.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A5x4.motherboard.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A5x4.motherboard.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

**FVP_VE_Cortex_A5x4.motherboard.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A5x4.motherboard.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A5x4.motherboard.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A5x4.motherboard.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A5x4.motherboard.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_VE_Cortex_A5x4.motherboard.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A5x4.motherboard.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A5x4.motherboard.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A5x4.motherboard.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A5x4.motherboard.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_VE_Cortex_A5x4.motherboard.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A5x4.motherboard.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_VE_Cortex_A5x4.motherboard.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A5x4.motherboard.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_VE_Cortex_A5x4.motherboard.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

**FVP_VE_Cortex_A5x4.motherboard.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A5x4.motherboard.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A5x4.motherboard.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A5x4.motherboard.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A5x4.motherboard.pl180_mci**

ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

**FVP_VE_Cortex_A5x4.motherboard.ps2keyboard**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A5x4.motherboard.ps2keyboard.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A5x4.motherboard.ps2keyboard.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A5x4.motherboard.ps2keyboard.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A5x4.motherboard.ps2keyboard.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A5x4.motherboard.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

**FVP_VE_Cortex_A5x4.motherboard.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A5x4.motherboard.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A5x4.motherboard.ps2mouse.ps2_clocktimer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A5x4.motherboard.ps2mouse.ps2_clocktimer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A5x4.motherboard.psram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A5x4.motherboard.smsc_91c111**
SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_VE_Cortex_A5x4.motherboard.sp805_wdog**
ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_VE_Cortex_A5x4.motherboard.sp805_wdog.clocktimer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A5x4.motherboard.sp805_wdog.clocktimer.timer**
A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A5x4.motherboard.sp805_wdog.clocktimer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A5x4.motherboard.sp805_wdog.clocktimer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A5x4.motherboard.sp810_sysctrl**

Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_VE_Cortex_A5x4.motherboard.sp810_sysctrl.clkdiv_clk0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A5x4.motherboard.sp810_sysctrl.clkdiv_clk1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A5x4.motherboard.sp810_sysctrl.clkdiv_clk2**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A5x4.motherboard.sp810_sysctrl.clkdiv_clk3**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A5x4.motherboard.terminal_0**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_A5x4.motherboard.terminal_1**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_A5x4.motherboard.terminal_2**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_A5x4.motherboard.terminal_3**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_A5x4.motherboard.ve_sysregs**

Type: `VE_SysRegs`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A5x4.motherboard.virtioblockdevice**

virtio block device.
Type: `VirtioBlockDevice`.

**FVP_VE_Cortex_A5x4.motherboard.virtiop9device**

virtio P9 server.
Type: `VirtioP9Device`.

**FVP_VE_Cortex_A5x4.motherboard.vis**

Display window for VE using Visualisation library.
Type: `VEVisualisation`.

**FVP_VE_Cortex_A5x4.motherboard.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_VE_Cortex_A5x4.motherboard.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A5x4.motherboard.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A5x4.motherboard.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A5x4.motherboard.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A5x4.motherboard.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A5x4.motherboard.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A5x4.motherboard.vram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A5x4.periph_clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

## 15.15 FVP_VE_Cortex-A7x1

List of instances in FVP_VE_Cortex-A7x1.

### FVP_VE_Cortex-A7x1 instances

**FVP_VE_Cortex_A7x1**

Top level component of the Cortex_A7x1 Versatile Express inspired model.
Type: `FVP_VE_Cortex_A7x1`.

**FVP_VE_Cortex_A7x1.cluster**

ARM Cortex-A7 Cluster CT model.
Type: `Cluster_ARM_Cortex-A7`.

**FVP_VE_Cortex_A7x1.cluster.cpu0**

ARM Cortex-A7 CT model.
Type: `ARM_Cortex-A7`.

**FVP_VE_Cortex_A7x1.cluster.cpu0.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_VE_Cortex_A7x1.cluster.cpu0.itlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_VE_Cortex_A7x1.cluster.cpu0.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A7x1.cluster.cpu0.l1icache**

PV Cache.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `PVCache`.

**FVP_VE_Cortex_A7x1.cluster.l2_cache**

PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A7x1.daughterboard**

Daughtercard, inspired by the ARM Versatile Express development platform.
Type: `VEDaughterBoard`.

**FVP_VE_Cortex_A7x1.daughterboard.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A7x1.daughterboard.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A7x1.daughterboard.dmc**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A7x1.daughterboard.dmc_phy**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A7x1.daughterboard.dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A7x1.daughterboard.dram_aliased**

Allow TrustZone secure/normal bus signals to be routed separately.
Type: `TZSwitch`.

**FVP_VE_Cortex_A7x1.daughterboard.dram_limit_4**

Allow TrustZone secure/normal bus signals to be routed separately.
Type: `TZSwitch`.

**FVP_VE_Cortex_A7x1.daughterboard.dram_limit_8**

Allow TrustZone secure/normal bus signals to be routed separately.
Type: `TZSwitch`.

**FVP_VE_Cortex_A7x1.daughterboard.exclusive_monitor**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A7x1.daughterboard.hdlcd**

ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).
Type: `PL370_HDLCD`.

**FVP_VE_Cortex_A7x1.daughterboard.hdlcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A7x1.daughterboard.hdlcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A7x1.daughterboard.hdlcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A7x1.daughterboard.hdlcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A7x1.daughterboard.introuter**

Interrupt Mapping peripheral (non-cascaded).
Type: `VEInterruptMapper`.

**FVP_VE_Cortex_A7x1.daughterboard.nonsecure_region**

Allow TrustZone secure/normal bus signals to be routed separately.
Type: `TZSwitch`.

**FVP_VE_Cortex_A7x1.daughterboard.secureDRAM**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A7x1.daughterboard.secureRO**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_VE_Cortex_A7x1.daughterboard.secureROloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A7x1.daughterboard.secureSRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A7x1.daughterboard.secure_region**
Allow TrustZone secure/normal bus signals to be routed separately.
Type: `TZSwitch`.

**FVP_VE_Cortex_A7x1.daughterboard.sram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A7x1.daughterboard.vedcc**
Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

**FVP_VE_Cortex_A7x1.globalcounter**
Memory Mapped Counter Module for Generic Timers.
Type: `MemoryMappedCounterModule`.

**FVP_VE_Cortex_A7x1.motherboard**
Model inspired by the ARM Versatile Express Motherboard.
Type: `VEMotherBoard`.

**FVP_VE_Cortex_A7x1.motherboard.Timer_0_1**
ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_VE_Cortex_A7x1.motherboard.Timer_0_1.clk_div0**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A7x1.motherboard.Timer_0_1.clk_div1**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A7x1.motherboard.Timer_0_1.counter0**
Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_VE_Cortex_A7x1.motherboard.Timer_0_1.counter1**
Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_VE_Cortex_A7x1.motherboard.Timer_2_3**
ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A7x1.motherboard.Timer_2_3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A7x1.motherboard.Timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A7x1.motherboard.Timer_2_3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_VE_Cortex_A7x1.motherboard.Timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_VE_Cortex_A7x1.motherboard.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**FVP_VE_Cortex_A7x1.motherboard.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A7x1.motherboard.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A7x1.motherboard.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A7x1.motherboard.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A7x1.motherboard.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A7x1.motherboard.dummy_local_dap_rom**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A7x1.motherboard.dummy_ram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A7x1.motherboard.dummy_usb**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A7x1.motherboard.flash0**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_VE_Cortex_A7x1.motherboard.flash1**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_VE_Cortex_A7x1.motherboard.flashloader0**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_VE_Cortex_A7x1.motherboard.flashloader1**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_VE_Cortex_A7x1.motherboard.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_VE_Cortex_A7x1.motherboard.mmc**

Generic Multimedia Card.
Type: `MMC`.

**FVP_VE_Cortex_A7x1.motherboard.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A7x1.motherboard.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A7x1.motherboard.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A7x1.motherboard.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A7x1.motherboard.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_A7x1.motherboard.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A7x1.motherboard.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A7x1.motherboard.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A7x1.motherboard.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A7x1.motherboard.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A7x1.motherboard.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_A7x1.motherboard.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A7x1.motherboard.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A7x1.motherboard.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A7x1.motherboard.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A7x1.motherboard.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A7x1.motherboard.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A7x1.motherboard.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A7x1.motherboard.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A7x1.motherboard.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A7x1.motherboard.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A7x1.motherboard.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A7x1.motherboard.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_A7x1.motherboard.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A7x1.motherboard.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A7x1.motherboard.pl011_uart3.timer.timer**
A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A7x1.motherboard.pl011_uart3.timer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A7x1.motherboard.pl011_uart3.timer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A7x1.motherboard.pl031_rtc**
ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

**FVP_VE_Cortex_A7x1.motherboard.pl031_rtc.timer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A7x1.motherboard.pl031_rtc.timer.timer**
A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A7x1.motherboard.pl031_rtc.timer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A7x1.motherboard.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A7x1.motherboard.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_VE_Cortex_A7x1.motherboard.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A7x1.motherboard.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A7x1.motherboard.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A7x1.motherboard.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A7x1.motherboard.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_VE_Cortex_A7x1.motherboard.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A7x1.motherboard.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A7x1.motherboard.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A7x1.motherboard.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_VE_Cortex_A7x1.motherboard.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

**FVP_VE_Cortex_A7x1.motherboard.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A7x1.motherboard.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A7x1.motherboard.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A7x1.motherboard.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A7x1.motherboard.pl180_mci**

ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

**FVP_VE_Cortex_A7x1.motherboard.ps2keyboard**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `PS2Keyboard`.

**FVP_VE_Cortex_A7x1.motherboard.ps2keyboard.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A7x1.motherboard.ps2keyboard.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A7x1.motherboard.ps2keyboard.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A7x1.motherboard.ps2keyboard.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A7x1.motherboard.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

**FVP_VE_Cortex_A7x1.motherboard.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A7x1.motherboard.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A7x1.motherboard.ps2mouse.ps2_clocktimer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A7x1.motherboard.ps2mouse.ps2_clocktimer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A7x1.motherboard.psram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A7x1.motherboard.smsc_91c111**
SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_VE_Cortex_A7x1.motherboard.sp805_wdog**
ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_VE_Cortex_A7x1.motherboard.sp805_wdog.clocktimer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A7x1.motherboard.sp805_wdog.clocktimer.timer**
A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A7x1.motherboard.sp805_wdog.clocktimer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A7x1.motherboard.sp805_wdog.clocktimer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A7x1.motherboard.sp810_sysctrl**

Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_VE_Cortex_A7x1.motherboard.sp810_sysctrl.clkdiv_clk0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A7x1.motherboard.sp810_sysctrl.clkdiv_clk1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A7x1.motherboard.sp810_sysctrl.clkdiv_clk2**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A7x1.motherboard.sp810_sysctrl.clkdiv_clk3**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A7x1.motherboard.terminal_0**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_A7x1.motherboard.terminal_1**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_A7x1.motherboard.terminal_2**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_A7x1.motherboard.terminal_3**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_A7x1.motherboard.ve_sysregs**

Type: `VE_SysRegs`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A7x1.motherboard.virtioblockdevice**

virtio block device.
Type: VirtioBlockDevice.

**FVP_VE_Cortex_A7x1.motherboard.virtiop9device**

virtio P9 server.
Type: VirtioP9Device.

**FVP_VE_Cortex_A7x1.motherboard.vis**

Display window for VE using Visualisation library.
Type: VEVisualisation.

**FVP_VE_Cortex_A7x1.motherboard.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: VisEventRecorder.

**FVP_VE_Cortex_A7x1.motherboard.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: ClockDivider.

**FVP_VE_Cortex_A7x1.motherboard.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: ClockTimerThread.

**FVP_VE_Cortex_A7x1.motherboard.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: ClockTimerThread64.

**FVP_VE_Cortex_A7x1.motherboard.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: SchedulerThread.

**FVP_VE_Cortex_A7x1.motherboard.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: SchedulerThreadEvent.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A7x1.motherboard.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A7x1.motherboard.vram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

# 15.16 FVP_VE_Cortex-A7x2

List of instances in FVP_VE_Cortex-A7x2.

## FVP_VE_Cortex-A7x2 instances

**FVP_VE_Cortex_A7x2**

Top level component of the Cortex_A7x2 Versatile Express inspired model.
Type: `FVP_VE_Cortex_A7x2`.

**FVP_VE_Cortex_A7x2.cluster**

ARM Cortex-A7 Cluster CT model.
Type: `Cluster_ARM_Cortex-A7`.

**FVP_VE_Cortex_A7x2.cluster.cpu0**

ARM Cortex-A7 CT model.
Type: `ARM_Cortex-A7`.

**FVP_VE_Cortex_A7x2.cluster.cpu0.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_VE_Cortex_A7x2.cluster.cpu0.itlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_VE_Cortex_A7x2.cluster.cpu0.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A7x2.cluster.cpu0.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A7x2.cluster.cpu1**

ARM Cortex-A7 CT model.
Type: `ARM_Cortex-A7`.

**FVP_VE_Cortex_A7x2.cluster.cpu1.dtlb**

TLB - instruction, data or unified.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `TlbCadi`.

**FVP_VE_Cortex_A7x2.cluster.cpu1.itlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_VE_Cortex_A7x2.cluster.cpu1.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A7x2.cluster.cpu1.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A7x2.cluster.l2_cache**

PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A7x2.daughterboard**

Daughtercard, inspired by the ARM Versatile Express development platform.
Type: `VEDaughterBoard`.

**FVP_VE_Cortex_A7x2.daughterboard.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A7x2.daughterboard.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A7x2.daughterboard.dmc**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A7x2.daughterboard.dmc_phy**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A7x2.daughterboard.dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A7x2.daughterboard.dram_aliased**

Allow TrustZone secure/normal bus signals to be routed separately.
Type: `TZSwitch`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A7x2.daughterboard.dram_limit_4**

Allow TrustZone secure/normal bus signals to be routed separately.
Type: `TZSwitch`.

**FVP_VE_Cortex_A7x2.daughterboard.dram_limit_8**

Allow TrustZone secure/normal bus signals to be routed separately.
Type: `TZSwitch`.

**FVP_VE_Cortex_A7x2.daughterboard.exclusive_monitor**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_VE_Cortex_A7x2.daughterboard.hdlcd**

ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).
Type: `PL370_HDLCD`.

**FVP_VE_Cortex_A7x2.daughterboard.hdlcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A7x2.daughterboard.hdlcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A7x2.daughterboard.hdlcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A7x2.daughterboard.hdlcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A7x2.daughterboard.introuter**

Interrupt Mapping peripheral (non-cascaded).
Type: `VEInterruptMapper`.

**FVP_VE_Cortex_A7x2.daughterboard.nonsecure_region**

Allow TrustZone secure/normal bus signals to be routed separately.
Type: `TZSwitch`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A7x2.daughterboard.secureDRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A7x2.daughterboard.secureRO**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_VE_Cortex_A7x2.daughterboard.secureROloader**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_VE_Cortex_A7x2.daughterboard.secureSRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A7x2.daughterboard.secure_region**
Allow TrustZone secure/normal bus signals to be routed separately.
Type: `TZSwitch`.

**FVP_VE_Cortex_A7x2.daughterboard.sram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A7x2.daughterboard.vedcc**
Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

**FVP_VE_Cortex_A7x2.globalcounter**
Memory Mapped Counter Module for Generic Timers.
Type: `MemoryMappedCounterModule`.

**FVP_VE_Cortex_A7x2.motherboard**
Model inspired by the ARM Versatile Express Motherboard.
Type: `VEMotherBoard`.

**FVP_VE_Cortex_A7x2.motherboard.Timer_0_1**
ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_VE_Cortex_A7x2.motherboard.Timer_0_1.clk_div0**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A7x2.motherboard.Timer_0_1.clk_div1**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A7x2.motherboard.Timer_0_1.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_VE_Cortex_A7x2.motherboard.Timer_0_1.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_VE_Cortex_A7x2.motherboard.Timer_2_3**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_VE_Cortex_A7x2.motherboard.Timer_2_3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A7x2.motherboard.Timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A7x2.motherboard.Timer_2_3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_VE_Cortex_A7x2.motherboard.Timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_VE_Cortex_A7x2.motherboard.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**FVP_VE_Cortex_A7x2.motherboard.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A7x2.motherboard.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A7x2.motherboard.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A7x2.motherboard.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A7x2.motherboard.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A7x2.motherboard.dummy_local_dap_rom**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A7x2.motherboard.dummy_ram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A7x2.motherboard.dummy_usb**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A7x2.motherboard.flash0**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_VE_Cortex_A7x2.motherboard.flash1**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_VE_Cortex_A7x2.motherboard.flashloader0**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_VE_Cortex_A7x2.motherboard.flashloader1**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_VE_Cortex_A7x2.motherboard.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A7x2.motherboard.mmc**

Generic Multimedia Card.
Type: `MMC`.

**FVP_VE_Cortex_A7x2.motherboard.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A7x2.motherboard.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A7x2.motherboard.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A7x2.motherboard.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A7x2.motherboard.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_A7x2.motherboard.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A7x2.motherboard.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ClockTimerThread`.

**FVP_VE_Cortex_A7x2.motherboard.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A7x2.motherboard.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A7x2.motherboard.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A7x2.motherboard.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_A7x2.motherboard.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A7x2.motherboard.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A7x2.motherboard.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A7x2.motherboard.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A7x2.motherboard.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A7x2.motherboard.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_A7x2.motherboard.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A7x2.motherboard.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A7x2.motherboard.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A7x2.motherboard.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A7x2.motherboard.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A7x2.motherboard.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A7x2.motherboard.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A7x2.motherboard.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A7x2.motherboard.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A7x2.motherboard.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A7x2.motherboard.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A7x2.motherboard.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

**FVP_VE_Cortex_A7x2.motherboard.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A7x2.motherboard.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A7x2.motherboard.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A7x2.motherboard.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A7x2.motherboard.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_VE_Cortex_A7x2.motherboard.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A7x2.motherboard.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A7x2.motherboard.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A7x2.motherboard.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A7x2.motherboard.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_VE_Cortex_A7x2.motherboard.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A7x2.motherboard.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_VE_Cortex_A7x2.motherboard.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A7x2.motherboard.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_VE_Cortex_A7x2.motherboard.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

**FVP_VE_Cortex_A7x2.motherboard.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A7x2.motherboard.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A7x2.motherboard.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `SchedulerThread`.

**FVP_VE_Cortex_A7x2.motherboard.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A7x2.motherboard.pl180_mci**

ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

**FVP_VE_Cortex_A7x2.motherboard.ps2keyboard**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

**FVP_VE_Cortex_A7x2.motherboard.ps2keyboard.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A7x2.motherboard.ps2keyboard.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A7x2.motherboard.ps2keyboard.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A7x2.motherboard.ps2keyboard.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A7x2.motherboard.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A7x2.motherboard.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A7x2.motherboard.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A7x2.motherboard.ps2mouse.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A7x2.motherboard.ps2mouse.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A7x2.motherboard.psram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A7x2.motherboard.smsc_91c111**

SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_VE_Cortex_A7x2.motherboard.sp805_wdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_VE_Cortex_A7x2.motherboard.sp805_wdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A7x2.motherboard.sp805_wdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A7x2.motherboard.sp805_wdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A7x2.motherboard.sp805_wdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A7x2.motherboard.sp810_sysctrl**

Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_VE_Cortex_A7x2.motherboard.sp810_sysctrl.clkdiv_clk0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A7x2.motherboard.sp810_sysctrl.clkdiv_clk1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A7x2.motherboard.sp810_sysctrl.clkdiv_clk2**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A7x2.motherboard.sp810_sysctrl.clkdiv_clk3**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A7x2.motherboard.terminal_0**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_A7x2.motherboard.terminal_1**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_A7x2.motherboard.terminal_2**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_A7x2.motherboard.terminal_3**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_A7x2.motherboard.ve_sysregs**

Type: `VE_SysRegs`.

**FVP_VE_Cortex_A7x2.motherboard.virtioblockdevice**

virtio block device.
Type: `VirtioBlockDevice`.

**FVP_VE_Cortex_A7x2.motherboard.virtiop9device**

virtio P9 server.
Type: `VirtioP9Device`.

**FVP_VE_Cortex_A7x2.motherboard.vis**

Display window for VE using Visualisation library.
Type: `VEVisualisation`.

**FVP_VE_Cortex_A7x2.motherboard.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_VE_Cortex_A7x2.motherboard.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A7x2.motherboard.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A7x2.motherboard.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A7x2.motherboard.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A7x2.motherboard.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A7x2.motherboard.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A7x2.motherboard.vram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

# 15.17 FVP_VE_Cortex-A7x4

List of instances in FVP_VE_Cortex-A7x4.

## FVP_VE_Cortex-A7x4 instances

**FVP_VE_Cortex_A7x4**

Top level component of the Cortex_A7x3 Versatile Express inspired model.
Type: `FVP_VE_Cortex_A7x4`.

**FVP_VE_Cortex_A7x4.cluster**

ARM Cortex-A7 Cluster CT model.
Type: `Cluster_ARM_Cortex-A7`.

**FVP_VE_Cortex_A7x4.cluster.cpu0**

ARM Cortex-A7 CT model.
Type: `ARM_Cortex-A7`.

**FVP_VE_Cortex_A7x4.cluster.cpu0.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A7x4.cluster.cpu0.itlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_VE_Cortex_A7x4.cluster.cpu0.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A7x4.cluster.cpu0.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A7x4.cluster.cpu1**
ARM Cortex-A7 CT model.
Type: `ARM_Cortex-A7`.

**FVP_VE_Cortex_A7x4.cluster.cpu1.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_VE_Cortex_A7x4.cluster.cpu1.itlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_VE_Cortex_A7x4.cluster.cpu1.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A7x4.cluster.cpu1.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A7x4.cluster.cpu2**
ARM Cortex-A7 CT model.
Type: `ARM_Cortex-A7`.

**FVP_VE_Cortex_A7x4.cluster.cpu2.dtlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_VE_Cortex_A7x4.cluster.cpu2.itlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_VE_Cortex_A7x4.cluster.cpu2.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A7x4.cluster.cpu2.l1icache**
PV Cache.
Type: `PVCache`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A7x4.cluster.cpu3**

ARM Cortex-A7 CT model.
Type: `ARM_Cortex-A7`.

**FVP_VE_Cortex_A7x4.cluster.cpu3.dtlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_VE_Cortex_A7x4.cluster.cpu3.itlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_VE_Cortex_A7x4.cluster.cpu3.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A7x4.cluster.cpu3.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A7x4.cluster.l2_cache**

PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A7x4.daughterboard**

Daughtercard, inspired by the ARM Versatile Express development platform.
Type: `VEDaughterBoard`.

**FVP_VE_Cortex_A7x4.daughterboard.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A7x4.daughterboard.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A7x4.daughterboard.dmc**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A7x4.daughterboard.dmc_phy**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A7x4.daughterboard.dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A7x4.daughterboard.dram_aliased**

Allow TrustZone secure/normal bus signals to be routed separately.
Type: `TZSwitch`.

**FVP_VE_Cortex_A7x4.daughterboard.dram_limit_4**

Allow TrustZone secure/normal bus signals to be routed separately.
Type: `TZSwitch`.

**FVP_VE_Cortex_A7x4.daughterboard.dram_limit_8**

Allow TrustZone secure/normal bus signals to be routed separately.
Type: `TZSwitch`.

**FVP_VE_Cortex_A7x4.daughterboard.exclusive_monitor**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_VE_Cortex_A7x4.daughterboard.hdlcd**

ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).
Type: `PL370_HDLCD`.

**FVP_VE_Cortex_A7x4.daughterboard.hdlcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A7x4.daughterboard.hdlcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A7x4.daughterboard.hdlcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A7x4.daughterboard.hdlcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A7x4.daughterboard.introuter**

Interrupt Mapping peripheral (non-cascaded).
Type: `VEInterruptMapper`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A7x4.daughterboard.nonsecure_region**
Allow TrustZone secure/normal bus signals to be routed separately.
Type: `TZSwitch`.

**FVP_VE_Cortex_A7x4.daughterboard.secureDRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A7x4.daughterboard.secureRO**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_VE_Cortex_A7x4.daughterboard.secureROloader**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_VE_Cortex_A7x4.daughterboard.secureSRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A7x4.daughterboard.secure_region**
Allow TrustZone secure/normal bus signals to be routed separately.
Type: `TZSwitch`.

**FVP_VE_Cortex_A7x4.daughterboard.sram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A7x4.daughterboard.vedcc**
Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

**FVP_VE_Cortex_A7x4.globalcounter**
Memory Mapped Counter Module for Generic Timers.
Type: `MemoryMappedCounterModule`.

**FVP_VE_Cortex_A7x4.motherboard**
Model inspired by the ARM Versatile Express Motherboard.
Type: `VEMotherBoard`.

**FVP_VE_Cortex_A7x4.motherboard.Timer_0_1**
ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_VE_Cortex_A7x4.motherboard.Timer_0_1.clk_div0**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A7x4.motherboard.Timer_0_1.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A7x4.motherboard.Timer_0_1.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_VE_Cortex_A7x4.motherboard.Timer_0_1.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_VE_Cortex_A7x4.motherboard.Timer_2_3**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_VE_Cortex_A7x4.motherboard.Timer_2_3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A7x4.motherboard.Timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A7x4.motherboard.Timer_2_3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_VE_Cortex_A7x4.motherboard.Timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_VE_Cortex_A7x4.motherboard.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**FVP_VE_Cortex_A7x4.motherboard.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A7x4.motherboard.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A7x4.motherboard.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A7x4.motherboard.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A7x4.motherboard.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A7x4.motherboard.dummy_local_dap_rom**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A7x4.motherboard.dummy_ram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A7x4.motherboard.dummy_usb**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A7x4.motherboard.flash0**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_VE_Cortex_A7x4.motherboard.flash1**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_VE_Cortex_A7x4.motherboard.flashloader0**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A7x4.motherboard.flashloader1**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_VE_Cortex_A7x4.motherboard.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_VE_Cortex_A7x4.motherboard.mmc**

Generic Multimedia Card.
Type: `MMC`.

**FVP_VE_Cortex_A7x4.motherboard.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A7x4.motherboard.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A7x4.motherboard.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A7x4.motherboard.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A7x4.motherboard.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_A7x4.motherboard.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A7x4.motherboard.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A7x4.motherboard.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A7x4.motherboard.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A7x4.motherboard.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A7x4.motherboard.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_A7x4.motherboard.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A7x4.motherboard.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A7x4.motherboard.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A7x4.motherboard.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A7x4.motherboard.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A7x4.motherboard.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_A7x4.motherboard.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A7x4.motherboard.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A7x4.motherboard.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A7x4.motherboard.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A7x4.motherboard.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A7x4.motherboard.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_A7x4.motherboard.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A7x4.motherboard.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A7x4.motherboard.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A7x4.motherboard.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A7x4.motherboard.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A7x4.motherboard.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

**FVP_VE_Cortex_A7x4.motherboard.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal()

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A7x4.motherboard.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A7x4.motherboard.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A7x4.motherboard.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A7x4.motherboard.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_VE_Cortex_A7x4.motherboard.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A7x4.motherboard.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A7x4.motherboard.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A7x4.motherboard.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A7x4.motherboard.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_VE_Cortex_A7x4.motherboard.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A7x4.motherboard.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_VE_Cortex_A7x4.motherboard.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A7x4.motherboard.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_VE_Cortex_A7x4.motherboard.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

**FVP_VE_Cortex_A7x4.motherboard.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A7x4.motherboard.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A7x4.motherboard.pl111_clcd.pl11x_clcd.timer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A7x4.motherboard.pl111_clcd.pl11x_clcd.timer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A7x4.motherboard.pl180_mci**
ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

**FVP_VE_Cortex_A7x4.motherboard.ps2keyboard**
Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

**FVP_VE_Cortex_A7x4.motherboard.ps2keyboard.ps2_clocktimer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A7x4.motherboard.ps2keyboard.ps2_clocktimer.timer**
A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A7x4.motherboard.ps2keyboard.ps2_clocktimer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A7x4.motherboard.ps2keyboard.ps2_clocktimer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A7x4.motherboard.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

**FVP_VE_Cortex_A7x4.motherboard.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A7x4.motherboard.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A7x4.motherboard.ps2mouse.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A7x4.motherboard.ps2mouse.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A7x4.motherboard.psram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A7x4.motherboard.smsc_91c111**

SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_VE_Cortex_A7x4.motherboard.sp805_wdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_VE_Cortex_A7x4.motherboard.sp805_wdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A7x4.motherboard.sp805_wdog.clocktimer.timer**
A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A7x4.motherboard.sp805_wdog.clocktimer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A7x4.motherboard.sp805_wdog.clocktimer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A7x4.motherboard.sp810_sysctrl**
Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_VE_Cortex_A7x4.motherboard.sp810_sysctrl.clkdiv_clk0**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A7x4.motherboard.sp810_sysctrl.clkdiv_clk1**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A7x4.motherboard.sp810_sysctrl.clkdiv_clk2**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A7x4.motherboard.sp810_sysctrl.clkdiv_clk3**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**`FVP_VE_Cortex_A7x4.motherboard.terminal_0`**
Telnet terminal interface.
Type: `TelnetTerminal`.

**`FVP_VE_Cortex_A7x4.motherboard.terminal_1`**
Telnet terminal interface.
Type: `TelnetTerminal`.

**`FVP_VE_Cortex_A7x4.motherboard.terminal_2`**
Telnet terminal interface.
Type: `TelnetTerminal`.

**`FVP_VE_Cortex_A7x4.motherboard.terminal_3`**
Telnet terminal interface.
Type: `TelnetTerminal`.

**`FVP_VE_Cortex_A7x4.motherboard.ve_sysregs`**
Type: `VE_SysRegs`.

**`FVP_VE_Cortex_A7x4.motherboard.virtioblockdevice`**
virtio block device.
Type: `VirtioBlockDevice`.

**`FVP_VE_Cortex_A7x4.motherboard.virtiop9device`**
virtio P9 server.
Type: `VirtioP9Device`.

**`FVP_VE_Cortex_A7x4.motherboard.vis`**
Display window for VE using Visualisation library.
Type: `VEVisualisation`.

**`FVP_VE_Cortex_A7x4.motherboard.vis.recorder`**
Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**`FVP_VE_Cortex_A7x4.motherboard.vis.recorder.playbackDivider`**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**`FVP_VE_Cortex_A7x4.motherboard.vis.recorder.playbackTimer`**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**`FVP_VE_Cortex_A7x4.motherboard.vis.recorder.playbackTimer.timer`**
A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**`FVP_VE_Cortex_A7x4.motherboard.vis.recorder.playbackTimer.timer.thread`**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**`FVP_VE_Cortex_A7x4.motherboard.vis.recorder.playbackTimer.timer.thread_event`**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**`FVP_VE_Cortex_A7x4.motherboard.vis.recorder.recordingDivider`**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**`FVP_VE_Cortex_A7x4.motherboard.vram`**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

# 15.18 FVP_VE_Cortex-A9x1

List of instances in FVP_VE_Cortex-A9x1.

## FVP_VE_Cortex-A9x1 instances

**`FVP_VE_Cortex_A9x1`**
Top level component of the Cortex-A9x1 Versatile Express inspired model.
Type: `FVP_VE_Cortex_A9x1`.

**`FVP_VE_Cortex_A9x1.cluster`**
ARM CORTEXA9MP Cluster CT model.
Type: `Cluster_ARM_Cortex-A9MP`.

**`FVP_VE_Cortex_A9x1.cluster.cpu0`**
ARM CORTEXA9MP CT model.
Type: `ARM_Cortex-A9MP`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A9x1.cluster.cpu0.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A9x1.cluster.cpu0.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A9x1.cluster.cpu0.utlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_VE_Cortex_A9x1.daughterboard**

Daughtercard, inspired by the ARM Versatile Express development platform.
Type: `VEDaughterBoard`.

**FVP_VE_Cortex_A9x1.daughterboard.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A9x1.daughterboard.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A9x1.daughterboard.dmc**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A9x1.daughterboard.dmc_phy**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A9x1.daughterboard.dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A9x1.daughterboard.dram_aliased**

Allow TrustZone secure/normal bus signals to be routed separately.
Type: `TZSwitch`.

**FVP_VE_Cortex_A9x1.daughterboard.dram_limit_4**

Allow TrustZone secure/normal bus signals to be routed separately.
Type: `TZSwitch`.

**FVP_VE_Cortex_A9x1.daughterboard.dram_limit_8**

Allow TrustZone secure/normal bus signals to be routed separately.
Type: `TZSwitch`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A9x1.daughterboard.exclusive_monitor**

Global exclusive monitor.
Type: PVBusExclusiveMonitor.

**FVP_VE_Cortex_A9x1.daughterboard.hdlcd**

ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).
Type: PL370_HDLCD.

**FVP_VE_Cortex_A9x1.daughterboard.hdlcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: ClockTimerThread.

**FVP_VE_Cortex_A9x1.daughterboard.hdlcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: ClockTimerThread64.

**FVP_VE_Cortex_A9x1.daughterboard.hdlcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: SchedulerThread.

**FVP_VE_Cortex_A9x1.daughterboard.hdlcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: SchedulerThreadEvent.

**FVP_VE_Cortex_A9x1.daughterboard.introuter**

Interrupt Mapping peripheral (non-cascaded).
Type: VEInterruptMapper.

**FVP_VE_Cortex_A9x1.daughterboard.nonsecure_region**

Allow TrustZone secure/normal bus signals to be routed separately.
Type: TZSwitch.

**FVP_VE_Cortex_A9x1.daughterboard.secureDRAM**

RAM device, can be dynamic or static ram.
Type: RAMDevice.

**FVP_VE_Cortex_A9x1.daughterboard.secureRO**

Intel Strata Flash J3 LISA+ model.
Type: IntelStrataFlashJ3.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A9x1.daughterboard.secureROloader**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_VE_Cortex_A9x1.daughterboard.secureSRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A9x1.daughterboard.secure_region**
Allow TrustZone secure/normal bus signals to be routed separately.
Type: `TZSwitch`.

**FVP_VE_Cortex_A9x1.daughterboard.sram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A9x1.daughterboard.vedcc**
Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

**FVP_VE_Cortex_A9x1.motherboard**
Model inspired by the ARM Versatile Express Motherboard.
Type: `VEMotherBoard`.

**FVP_VE_Cortex_A9x1.motherboard.Timer_0_1**
ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_VE_Cortex_A9x1.motherboard.Timer_0_1.clk_div0**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A9x1.motherboard.Timer_0_1.clk_div1**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A9x1.motherboard.Timer_0_1.counter0**
Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_VE_Cortex_A9x1.motherboard.Timer_0_1.counter1**
Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_VE_Cortex_A9x1.motherboard.Timer_2_3**
ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A9x1.motherboard.Timer_2_3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A9x1.motherboard.Timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A9x1.motherboard.Timer_2_3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_VE_Cortex_A9x1.motherboard.Timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_VE_Cortex_A9x1.motherboard.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**FVP_VE_Cortex_A9x1.motherboard.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A9x1.motherboard.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A9x1.motherboard.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A9x1.motherboard.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A9x1.motherboard.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A9x1.motherboard.dummy_local_dap_rom**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A9x1.motherboard.dummy_ram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A9x1.motherboard.dummy_usb**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A9x1.motherboard.flash0**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_VE_Cortex_A9x1.motherboard.flash1**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_VE_Cortex_A9x1.motherboard.flashloader0**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_VE_Cortex_A9x1.motherboard.flashloader1**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_VE_Cortex_A9x1.motherboard.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_VE_Cortex_A9x1.motherboard.mmc**

Generic Multimedia Card.
Type: `MMC`.

**FVP_VE_Cortex_A9x1.motherboard.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A9x1.motherboard.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A9x1.motherboard.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A9x1.motherboard.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A9x1.motherboard.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_A9x1.motherboard.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A9x1.motherboard.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A9x1.motherboard.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A9x1.motherboard.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A9x1.motherboard.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A9x1.motherboard.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_A9x1.motherboard.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A9x1.motherboard.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A9x1.motherboard.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A9x1.motherboard.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A9x1.motherboard.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A9x1.motherboard.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A9x1.motherboard.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A9x1.motherboard.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A9x1.motherboard.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A9x1.motherboard.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A9x1.motherboard.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A9x1.motherboard.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_A9x1.motherboard.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A9x1.motherboard.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A9x1.motherboard.pl011_uart3.timer.timer**
A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A9x1.motherboard.pl011_uart3.timer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A9x1.motherboard.pl011_uart3.timer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A9x1.motherboard.pl031_rtc**
ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

**FVP_VE_Cortex_A9x1.motherboard.pl031_rtc.timer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A9x1.motherboard.pl031_rtc.timer.timer**
A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A9x1.motherboard.pl031_rtc.timer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A9x1.motherboard.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A9x1.motherboard.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_VE_Cortex_A9x1.motherboard.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A9x1.motherboard.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A9x1.motherboard.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A9x1.motherboard.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A9x1.motherboard.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_VE_Cortex_A9x1.motherboard.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A9x1.motherboard.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A9x1.motherboard.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A9x1.motherboard.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_VE_Cortex_A9x1.motherboard.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

**FVP_VE_Cortex_A9x1.motherboard.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A9x1.motherboard.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A9x1.motherboard.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A9x1.motherboard.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A9x1.motherboard.pl180_mci**

ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

**FVP_VE_Cortex_A9x1.motherboard.ps2keyboard**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `PS2Keyboard`.

**FVP_VE_Cortex_A9x1.motherboard.ps2keyboard.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A9x1.motherboard.ps2keyboard.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A9x1.motherboard.ps2keyboard.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A9x1.motherboard.ps2keyboard.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A9x1.motherboard.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

**FVP_VE_Cortex_A9x1.motherboard.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A9x1.motherboard.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A9x1.motherboard.ps2mouse.ps2_clocktimer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A9x1.motherboard.ps2mouse.ps2_clocktimer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A9x1.motherboard.psram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A9x1.motherboard.smsc_91c111**
SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_VE_Cortex_A9x1.motherboard.sp805_wdog**
ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_VE_Cortex_A9x1.motherboard.sp805_wdog.clocktimer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A9x1.motherboard.sp805_wdog.clocktimer.timer**
A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A9x1.motherboard.sp805_wdog.clocktimer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A9x1.motherboard.sp805_wdog.clocktimer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A9x1.motherboard.sp810_sysctrl**

Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_VE_Cortex_A9x1.motherboard.sp810_sysctrl.clkdiv_clk0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A9x1.motherboard.sp810_sysctrl.clkdiv_clk1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A9x1.motherboard.sp810_sysctrl.clkdiv_clk2**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A9x1.motherboard.sp810_sysctrl.clkdiv_clk3**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A9x1.motherboard.terminal_0**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_A9x1.motherboard.terminal_1**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_A9x1.motherboard.terminal_2**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_A9x1.motherboard.terminal_3**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_A9x1.motherboard.ve_sysregs**

Type: `VE_SysRegs`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A9x1.motherboard.virtioblockdevice**

virtio block device.
Type: `VirtioBlockDevice`.

**FVP_VE_Cortex_A9x1.motherboard.virtiop9device**

virtio P9 server.
Type: `VirtioP9Device`.

**FVP_VE_Cortex_A9x1.motherboard.vis**

Display window for VE using Visualisation library.
Type: `VEVisualisation`.

**FVP_VE_Cortex_A9x1.motherboard.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_VE_Cortex_A9x1.motherboard.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A9x1.motherboard.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A9x1.motherboard.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A9x1.motherboard.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A9x1.motherboard.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A9x1.motherboard.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A9x1.motherboard.vram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A9x1.periph_clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

## 15.19 FVP_VE_Cortex-A9x2

List of instances in FVP_VE_Cortex-A9x2.

### FVP_VE_Cortex-A9x2 instances

**FVP_VE_Cortex_A9x2**

Top level component of the Cortex-A9x2 Versatile Express inspired model.
Type: `FVP_VE_Cortex_A9x2`.

**FVP_VE_Cortex_A9x2.cluster**

ARM CORTEXA9MP Cluster CT model.
Type: `Cluster_ARM_Cortex-A9MP`.

**FVP_VE_Cortex_A9x2.cluster.cpu0**

ARM CORTEXA9MP CT model.
Type: `ARM_Cortex-A9MP`.

**FVP_VE_Cortex_A9x2.cluster.cpu0.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A9x2.cluster.cpu0.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A9x2.cluster.cpu0.utlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_VE_Cortex_A9x2.cluster.cpu1**

ARM CORTEXA9MP CT model.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ARM_Cortex-A9MP`.

**FVP_VE_Cortex_A9x2.cluster.cpu1.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A9x2.cluster.cpu1.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A9x2.cluster.cpu1.utlb**

TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_VE_Cortex_A9x2.daughterboard**

Daughtercard, inspired by the ARM Versatile Express development platform.
Type: `VEDaughterBoard`.

**FVP_VE_Cortex_A9x2.daughterboard.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A9x2.daughterboard.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A9x2.daughterboard.dmc**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A9x2.daughterboard.dmc_phy**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A9x2.daughterboard.dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A9x2.daughterboard.dram_aliased**

Allow TrustZone secure/normal bus signals to be routed separately.
Type: `TZSwitch`.

**FVP_VE_Cortex_A9x2.daughterboard.dram_limit_4**

Allow TrustZone secure/normal bus signals to be routed separately.
Type: `TZSwitch`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A9x2.daughterboard.dram_limit_8**

Allow TrustZone secure/normal bus signals to be routed separately.
Type: `TZSwitch`.

**FVP_VE_Cortex_A9x2.daughterboard.exclusive_monitor**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_VE_Cortex_A9x2.daughterboard.hdlcd**

ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).
Type: `PL370_HDLCD`.

**FVP_VE_Cortex_A9x2.daughterboard.hdlcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A9x2.daughterboard.hdlcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A9x2.daughterboard.hdlcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A9x2.daughterboard.hdlcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A9x2.daughterboard.introuter**

Interrupt Mapping peripheral (non-cascaded).
Type: `VEInterruptMapper`.

**FVP_VE_Cortex_A9x2.daughterboard.nonsecure_region**

Allow TrustZone secure/normal bus signals to be routed separately.
Type: `TZSwitch`.

**FVP_VE_Cortex_A9x2.daughterboard.secureDRAM**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A9x2.daughterboard.secureRO**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_VE_Cortex_A9x2.daughterboard.secureROloader**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_VE_Cortex_A9x2.daughterboard.secureSRAM**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A9x2.daughterboard.secure_region**
Allow TrustZone secure/normal bus signals to be routed separately.
Type: `TZSwitch`.

**FVP_VE_Cortex_A9x2.daughterboard.sram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A9x2.daughterboard.vedcc**
Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

**FVP_VE_Cortex_A9x2.motherboard**
Model inspired by the ARM Versatile Express Motherboard.
Type: `VEMotherBoard`.

**FVP_VE_Cortex_A9x2.motherboard.Timer_0_1**
ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_VE_Cortex_A9x2.motherboard.Timer_0_1.clk_div0**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A9x2.motherboard.Timer_0_1.clk_div1**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A9x2.motherboard.Timer_0_1.counter0**
Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_VE_Cortex_A9x2.motherboard.Timer_0_1.counter1**
Internal component used by SP804 Timer module.
Type: `CounterModule`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A9x2.motherboard.Timer_2_3**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_VE_Cortex_A9x2.motherboard.Timer_2_3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A9x2.motherboard.Timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A9x2.motherboard.Timer_2_3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_VE_Cortex_A9x2.motherboard.Timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_VE_Cortex_A9x2.motherboard.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**FVP_VE_Cortex_A9x2.motherboard.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A9x2.motherboard.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A9x2.motherboard.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A9x2.motherboard.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A9x2.motherboard.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A9x2.motherboard.dummy_local_dap_rom**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A9x2.motherboard.dummy_ram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A9x2.motherboard.dummy_usb**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A9x2.motherboard.flash0**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_VE_Cortex_A9x2.motherboard.flash1**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_VE_Cortex_A9x2.motherboard.flashloader0**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_VE_Cortex_A9x2.motherboard.flashloader1**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_VE_Cortex_A9x2.motherboard.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_VE_Cortex_A9x2.motherboard.mmc**

Generic Multimedia Card.
Type: `MMC`.

**FVP_VE_Cortex_A9x2.motherboard.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal()

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A9x2.motherboard.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A9x2.motherboard.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A9x2.motherboard.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A9x2.motherboard.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_A9x2.motherboard.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A9x2.motherboard.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A9x2.motherboard.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A9x2.motherboard.pl011_uart0.timer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A9x2.motherboard.pl011_uart0.timer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A9x2.motherboard.pl011_uart1**
ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_A9x2.motherboard.pl011_uart1.clk_divider**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A9x2.motherboard.pl011_uart1.timer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A9x2.motherboard.pl011_uart1.timer.timer**
A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A9x2.motherboard.pl011_uart1.timer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A9x2.motherboard.pl011_uart1.timer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A9x2.motherboard.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_A9x2.motherboard.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A9x2.motherboard.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A9x2.motherboard.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A9x2.motherboard.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A9x2.motherboard.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A9x2.motherboard.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_A9x2.motherboard.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A9x2.motherboard.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A9x2.motherboard.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A9x2.motherboard.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A9x2.motherboard.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A9x2.motherboard.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

**FVP_VE_Cortex_A9x2.motherboard.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A9x2.motherboard.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A9x2.motherboard.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A9x2.motherboard.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A9x2.motherboard.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_VE_Cortex_A9x2.motherboard.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A9x2.motherboard.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A9x2.motherboard.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A9x2.motherboard.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A9x2.motherboard.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_VE_Cortex_A9x2.motherboard.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A9x2.motherboard.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_VE_Cortex_A9x2.motherboard.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A9x2.motherboard.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_VE_Cortex_A9x2.motherboard.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

**FVP_VE_Cortex_A9x2.motherboard.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A9x2.motherboard.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A9x2.motherboard.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A9x2.motherboard.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A9x2.motherboard.pl180_mci**

ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A9x2.motherboard.ps2keyboard**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

**FVP_VE_Cortex_A9x2.motherboard.ps2keyboard.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A9x2.motherboard.ps2keyboard.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A9x2.motherboard.ps2keyboard.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A9x2.motherboard.ps2keyboard.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A9x2.motherboard.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

**FVP_VE_Cortex_A9x2.motherboard.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A9x2.motherboard.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A9x2.motherboard.ps2mouse.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A9x2.motherboard.ps2mouse.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A9x2.motherboard.psram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A9x2.motherboard.smsc_91c111**

SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_VE_Cortex_A9x2.motherboard.sp805_wdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_VE_Cortex_A9x2.motherboard.sp805_wdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A9x2.motherboard.sp805_wdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A9x2.motherboard.sp805_wdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `SchedulerThread`.

**FVP_VE_Cortex_A9x2.motherboard.sp805_wdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A9x2.motherboard.sp810_sysctrl**

Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_VE_Cortex_A9x2.motherboard.sp810_sysctrl.clkdiv_clk0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A9x2.motherboard.sp810_sysctrl.clkdiv_clk1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A9x2.motherboard.sp810_sysctrl.clkdiv_clk2**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A9x2.motherboard.sp810_sysctrl.clkdiv_clk3**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A9x2.motherboard.terminal_0**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_A9x2.motherboard.terminal_1**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_A9x2.motherboard.terminal_2**

Telnet terminal interface.
Type: `TelnetTerminal`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A9x2.motherboard.terminal_3**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_A9x2.motherboard.ve_sysregs**

Type: `VE_SysRegs`.

**FVP_VE_Cortex_A9x2.motherboard.virtioblockdevice**

virtio block device.
Type: `VirtioBlockDevice`.

**FVP_VE_Cortex_A9x2.motherboard.virtiop9device**

virtio P9 server.
Type: `VirtioP9Device`.

**FVP_VE_Cortex_A9x2.motherboard.vis**

Display window for VE using Visualisation library.
Type: `VEVisualisation`.

**FVP_VE_Cortex_A9x2.motherboard.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_VE_Cortex_A9x2.motherboard.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A9x2.motherboard.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A9x2.motherboard.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A9x2.motherboard.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `SchedulerThread`.

**`FVP_VE_Cortex_A9x2.motherboard.vis.recorder.playbackTimer.timer.thread_event`**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**`FVP_VE_Cortex_A9x2.motherboard.vis.recorder.recordingDivider`**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**`FVP_VE_Cortex_A9x2.motherboard.vram`**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**`FVP_VE_Cortex_A9x2.periph_clockdivider`**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

## 15.20 FVP_VE_Cortex-A9x4

List of instances in FVP_VE_Cortex-A9x4.

### FVP_VE_Cortex-A9x4 instances

**`FVP_VE_Cortex_A9x4`**

Top level component of the Cortex-A9x4 Versatile Express inspired model.
Type: `FVP_VE_Cortex_A9x4`.

**`FVP_VE_Cortex_A9x4.cluster`**

ARM CORTEXA9MP Cluster CT model.
Type: `Cluster_ARM_Cortex-A9MP`.

**`FVP_VE_Cortex_A9x4.cluster.cpu0`**

ARM CORTEXA9MP CT model.
Type: `ARM_Cortex-A9MP`.

**`FVP_VE_Cortex_A9x4.cluster.cpu0.l1dcache`**

PV Cache.
Type: `PVCache`.

**`FVP_VE_Cortex_A9x4.cluster.cpu0.l1icache`**

PV Cache.
Type: `PVCache`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A9x4.cluster.cpu0.utlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_VE_Cortex_A9x4.cluster.cpu1**
ARM CORTEXA9MP CT model.
Type: `ARM_Cortex-A9MP`.

**FVP_VE_Cortex_A9x4.cluster.cpu1.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A9x4.cluster.cpu1.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A9x4.cluster.cpu1.utlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_VE_Cortex_A9x4.cluster.cpu2**
ARM CORTEXA9MP CT model.
Type: `ARM_Cortex-A9MP`.

**FVP_VE_Cortex_A9x4.cluster.cpu2.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A9x4.cluster.cpu2.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A9x4.cluster.cpu2.utlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

**FVP_VE_Cortex_A9x4.cluster.cpu3**
ARM CORTEXA9MP CT model.
Type: `ARM_Cortex-A9MP`.

**FVP_VE_Cortex_A9x4.cluster.cpu3.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A9x4.cluster.cpu3.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_A9x4.cluster.cpu3.utlb**
TLB - instruction, data or unified.
Type: `TlbCadi`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A9x4.daughterboard**

Daughtercard, inspired by the ARM Versatile Express development platform.
Type: `VEDaughterBoard`.

**FVP_VE_Cortex_A9x4.daughterboard.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A9x4.daughterboard.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A9x4.daughterboard.dmc**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A9x4.daughterboard.dmc_phy**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A9x4.daughterboard.dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A9x4.daughterboard.dram_aliased**

Allow TrustZone secure/normal bus signals to be routed separately.
Type: `TZSwitch`.

**FVP_VE_Cortex_A9x4.daughterboard.dram_limit_4**

Allow TrustZone secure/normal bus signals to be routed separately.
Type: `TZSwitch`.

**FVP_VE_Cortex_A9x4.daughterboard.dram_limit_8**

Allow TrustZone secure/normal bus signals to be routed separately.
Type: `TZSwitch`.

**FVP_VE_Cortex_A9x4.daughterboard.exclusive_monitor**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_VE_Cortex_A9x4.daughterboard.hdlcd**

ARM PrimeCell HD Color LCD Controller (Nominal Designation PL370).
Type: `PL370_HDLCD`.

**FVP_VE_Cortex_A9x4.daughterboard.hdlcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal()

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A9x4.daughterboard.hdlcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A9x4.daughterboard.hdlcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A9x4.daughterboard.hdlcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A9x4.daughterboard.introuter**

Interrupt Mapping peripheral (non-cascaded).
Type: `VEInterruptMapper`.

**FVP_VE_Cortex_A9x4.daughterboard.nonsecure_region**

Allow TrustZone secure/normal bus signals to be routed separately.
Type: `TZSwitch`.

**FVP_VE_Cortex_A9x4.daughterboard.secureDRAM**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A9x4.daughterboard.secureRO**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_VE_Cortex_A9x4.daughterboard.secureROloader**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_VE_Cortex_A9x4.daughterboard.secureSRAM**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A9x4.daughterboard.secure_region**

Allow TrustZone secure/normal bus signals to be routed separately.
Type: `TZSwitch`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A9x4.daughterboard.sram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A9x4.daughterboard.vedcc**
Daughterboard Configuration Control (DCC).
Type: `VEDCC`.

**FVP_VE_Cortex_A9x4.motherboard**
Model inspired by the ARM Versatile Express Motherboard.
Type: `VEMotherBoard`.

**FVP_VE_Cortex_A9x4.motherboard.Timer_0_1**
ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_VE_Cortex_A9x4.motherboard.Timer_0_1.clk_div0**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A9x4.motherboard.Timer_0_1.clk_div1**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A9x4.motherboard.Timer_0_1.counter0**
Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_VE_Cortex_A9x4.motherboard.Timer_0_1.counter1**
Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_VE_Cortex_A9x4.motherboard.Timer_2_3**
ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_VE_Cortex_A9x4.motherboard.Timer_2_3.clk_div0**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A9x4.motherboard.Timer_2_3.clk_div1**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**`FVP_VE_Cortex_A9x4.motherboard.Timer_2_3.counter0`**
Internal component used by SP804 Timer module.
Type: `CounterModule`.

**`FVP_VE_Cortex_A9x4.motherboard.Timer_2_3.counter1`**
Internal component used by SP804 Timer module.
Type: `CounterModule`.

**`FVP_VE_Cortex_A9x4.motherboard.audioout`**
SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**`FVP_VE_Cortex_A9x4.motherboard.clock100Hz`**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**`FVP_VE_Cortex_A9x4.motherboard.clock24MHz`**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**`FVP_VE_Cortex_A9x4.motherboard.clock35MHz`**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**`FVP_VE_Cortex_A9x4.motherboard.clock50Hz`**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**`FVP_VE_Cortex_A9x4.motherboard.clockCLCD`**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A9x4.motherboard.dummy_local_dap_rom**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A9x4.motherboard.dummy_ram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A9x4.motherboard.dummy_usb**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A9x4.motherboard.flash0**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_VE_Cortex_A9x4.motherboard.flash1**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_VE_Cortex_A9x4.motherboard.flashloader0**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_VE_Cortex_A9x4.motherboard.flashloader1**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_VE_Cortex_A9x4.motherboard.hostbridge**
Host Socket Interface Component.
Type: `HostBridge`.

**FVP_VE_Cortex_A9x4.motherboard.mmc**
Generic Multimedia Card.
Type: `MMC`.

**FVP_VE_Cortex_A9x4.motherboard.mmc.timer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A9x4.motherboard.mmc.timer.timer**
A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A9x4.motherboard.mmc.timer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A9x4.motherboard.mmc.timer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A9x4.motherboard.pl011_uart0**
ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_A9x4.motherboard.pl011_uart0.clk_divider**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A9x4.motherboard.pl011_uart0.timer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A9x4.motherboard.pl011_uart0.timer.timer**
A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A9x4.motherboard.pl011_uart0.timer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A9x4.motherboard.pl011_uart0.timer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A9x4.motherboard.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_A9x4.motherboard.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A9x4.motherboard.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A9x4.motherboard.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A9x4.motherboard.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A9x4.motherboard.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A9x4.motherboard.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_A9x4.motherboard.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A9x4.motherboard.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A9x4.motherboard.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A9x4.motherboard.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A9x4.motherboard.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A9x4.motherboard.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_A9x4.motherboard.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A9x4.motherboard.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A9x4.motherboard.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A9x4.motherboard.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A9x4.motherboard.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A9x4.motherboard.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

**FVP_VE_Cortex_A9x4.motherboard.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A9x4.motherboard.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A9x4.motherboard.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A9x4.motherboard.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A9x4.motherboard.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_VE_Cortex_A9x4.motherboard.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A9x4.motherboard.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A9x4.motherboard.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A9x4.motherboard.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A9x4.motherboard.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_VE_Cortex_A9x4.motherboard.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A9x4.motherboard.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_VE_Cortex_A9x4.motherboard.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A9x4.motherboard.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_VE_Cortex_A9x4.motherboard.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

**FVP_VE_Cortex_A9x4.motherboard.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A9x4.motherboard.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A9x4.motherboard.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A9x4.motherboard.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A9x4.motherboard.pl180_mci**

ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

**FVP_VE_Cortex_A9x4.motherboard.ps2keyboard**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A9x4.motherboard.ps2keyboard.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A9x4.motherboard.ps2keyboard.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A9x4.motherboard.ps2keyboard.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A9x4.motherboard.ps2keyboard.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A9x4.motherboard.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

**FVP_VE_Cortex_A9x4.motherboard.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A9x4.motherboard.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A9x4.motherboard.ps2mouse.ps2_clocktimer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A9x4.motherboard.ps2mouse.ps2_clocktimer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_A9x4.motherboard.psram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A9x4.motherboard.smsc_91c111**
SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_VE_Cortex_A9x4.motherboard.sp805_wdog**
ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_VE_Cortex_A9x4.motherboard.sp805_wdog.clocktimer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A9x4.motherboard.sp805_wdog.clocktimer.timer**
A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A9x4.motherboard.sp805_wdog.clocktimer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A9x4.motherboard.sp805_wdog.clocktimer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A9x4.motherboard.sp810_sysctrl**

Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_VE_Cortex_A9x4.motherboard.sp810_sysctrl.clkdiv_clk0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A9x4.motherboard.sp810_sysctrl.clkdiv_clk1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A9x4.motherboard.sp810_sysctrl.clkdiv_clk2**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A9x4.motherboard.sp810_sysctrl.clkdiv_clk3**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A9x4.motherboard.terminal_0**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_A9x4.motherboard.terminal_1**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_A9x4.motherboard.terminal_2**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_A9x4.motherboard.terminal_3**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_A9x4.motherboard.ve_sysregs**

Type: `VE_SysRegs`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A9x4.motherboard.virtioblockdevice**

virtio block device.
Type: `VirtioBlockDevice`.

**FVP_VE_Cortex_A9x4.motherboard.virtiop9device**

virtio P9 server.
Type: `VirtioP9Device`.

**FVP_VE_Cortex_A9x4.motherboard.vis**

Display window for VE using Visualisation library.
Type: `VEVisualisation`.

**FVP_VE_Cortex_A9x4.motherboard.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_VE_Cortex_A9x4.motherboard.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A9x4.motherboard.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_A9x4.motherboard.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_A9x4.motherboard.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_A9x4.motherboard.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_A9x4.motherboard.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_A9x4.motherboard.vram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_A9x4.periph_clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

## 15.21 FVP_VE_Cortex-R4

List of instances in FVP_VE_Cortex-R4.

### FVP_VE_Cortex-R4 instances

**FVP_VE_Cortex_R4**

Top level component of the Cortex_R4 Versatile Express inspired model.
Type: `FVP_VE_Cortex_R4`.

**FVP_VE_Cortex_R4.daughterboard**

Cortex-R4 DaughterBoard for Versatile Express.
Type: `VEDaughterBoardCortex_R4`.

**FVP_VE_Cortex_R4.daughterboard.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R4.daughterboard.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R4.daughterboard.core**

ARM CORTEXR4 CT model.
Type: `ARM_Cortex-R4`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_R4.daughterboard.core.cpu0.l1dcache**

PV Cache.
Type: PVCache.

**FVP_VE_Cortex_R4.daughterboard.core.cpu0.l1icache**

PV Cache.
Type: PVCache.

**FVP_VE_Cortex_R4.daughterboard.dram**

RAM device, can be dynamic or static ram.
Type: RAMDevice.

**FVP_VE_Cortex_R4.daughterboard.exclusive_monitor**

Global exclusive monitor.
Type: PVBusExclusiveMonitor.

**FVP_VE_Cortex_R4.daughterboard.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: PL111_CLCD.

**FVP_VE_Cortex_R4.daughterboard.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: PL11x_CLCD.

**FVP_VE_Cortex_R4.daughterboard.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: ClockTimerThread.

**FVP_VE_Cortex_R4.daughterboard.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: ClockTimerThread64.

**FVP_VE_Cortex_R4.daughterboard.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: SchedulerThread.

**FVP_VE_Cortex_R4.daughterboard.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: SchedulerThreadEvent.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_R4.daughterboard.pl310_l2cc**
ARM PrimeCell Level 2 Cache Controller (PL310).
Type: `PL310_L2CC`.

**FVP_VE_Cortex_R4.daughterboard.pl390_gic**
Generic Interrupt Controller (PL390).
Type: `PL390_GIC`.

**FVP_VE_Cortex_R4.daughterboard.veinterruptmapper**
Interrupt Mapping peripheral (non-cascaded).
Type: `VEInterruptMapper`.

**FVP_VE_Cortex_R4.vemotherboard**
Model inspired by the ARM Versatile Express Motherboard for R profile.
Type: `VEMotherBoardR`.

**FVP_VE_Cortex_R4.vemotherboard.audioout**
SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**FVP_VE_Cortex_R4.vemotherboard.clock100Hz**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R4.vemotherboard.clock24MHz**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R4.vemotherboard.clock35MHz**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R4.vemotherboard.clock50Hz**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R4.vemotherboard.clockCLCD**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R4.vemotherboard.dummy_ram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_R4.vemotherboard.dummy_usb**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_R4.vemotherboard.flash0**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_VE_Cortex_R4.vemotherboard.flash1**
Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_VE_Cortex_R4.vemotherboard.flashloader0**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_VE_Cortex_R4.vemotherboard.flashloader1**
A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_VE_Cortex_R4.vemotherboard.hostbridge**
Host Socket Interface Component.
Type: `HostBridge`.

**FVP_VE_Cortex_R4.vemotherboard.mmc**
Generic Multimedia Card.
Type: `MMC`.

**FVP_VE_Cortex_R4.vemotherboard.mmc.timer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_R4.vemotherboard.mmc.timer.timer**
A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_R4.vemotherboard.mmc.timer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_R4.vemotherboard.mmc.timer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_R4.vemotherboard.pl011_uart0**
ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_R4.vemotherboard.pl011_uart0.clk_divider**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R4.vemotherboard.pl011_uart0.timer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_R4.vemotherboard.pl011_uart0.timer.timer**
A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_R4.vemotherboard.pl011_uart0.timer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_R4.vemotherboard.pl011_uart0.timer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_R4.vemotherboard.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_R4.vemotherboard.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R4.vemotherboard.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_R4.vemotherboard.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_R4.vemotherboard.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_R4.vemotherboard.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_R4.vemotherboard.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_R4.vemotherboard.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_R4.vemotherboard.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_R4.vemotherboard.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_R4.vemotherboard.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_R4.vemotherboard.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_R4.vemotherboard.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_R4.vemotherboard.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R4.vemotherboard.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_R4.vemotherboard.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_R4.vemotherboard.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_R4.vemotherboard.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_R4.vemotherboard.pl011_uart4**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_R4.vemotherboard.pl011_uart4.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R4.vemotherboard.pl011_uart4.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_R4.vemotherboard.pl011_uart4.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_R4.vemotherboard.pl011_uart4.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_R4.vemotherboard.pl011_uart4.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_R4.vemotherboard.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

**FVP_VE_Cortex_R4.vemotherboard.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_R4.vemotherboard.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_R4.vemotherboard.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_R4.vemotherboard.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_R4.vemotherboard.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_VE_Cortex_R4.vemotherboard.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_R4.vemotherboard.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_R4.vemotherboard.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_R4.vemotherboard.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_R4.vemotherboard.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_VE_Cortex_R4.vemotherboard.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R4.vemotherboard.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_VE_Cortex_R4.vemotherboard.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R4.vemotherboard.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_VE_Cortex_R4.vemotherboard.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

**FVP_VE_Cortex_R4.vemotherboard.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_R4.vemotherboard.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_R4.vemotherboard.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_R4.vemotherboard.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_R4.vemotherboard.pl180_mci**

ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

**FVP_VE_Cortex_R4.vemotherboard.ps2keyboard**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

**FVP_VE_Cortex_R4.vemotherboard.ps2keyboard.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_R4.vemotherboard.ps2keyboard.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ClockTimerThread64`.

**FVP_VE_Cortex_R4.vemotherboard.ps2keyboard.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_R4.vemotherboard.ps2keyboard.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_R4.vemotherboard.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

**FVP_VE_Cortex_R4.vemotherboard.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_R4.vemotherboard.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_R4.vemotherboard.ps2mouse.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_R4.vemotherboard.ps2mouse.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_R4.vemotherboard.psram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_R4.vemotherboard.smsc_91c111**

SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_R4.vemotherboard.sp805_wdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_VE_Cortex_R4.vemotherboard.sp805_wdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_R4.vemotherboard.sp805_wdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_R4.vemotherboard.sp805_wdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_R4.vemotherboard.sp805_wdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_R4.vemotherboard.sp810_sysctrl**

Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_VE_Cortex_R4.vemotherboard.sp810_sysctrl.clkdiv_clk0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R4.vemotherboard.sp810_sysctrl.clkdiv_clk1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_R4.vemotherboard.sp810_sysctrl.clkdiv_clk2**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R4.vemotherboard.sp810_sysctrl.clkdiv_clk3**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R4.vemotherboard.terminal_0**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_R4.vemotherboard.terminal_1**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_R4.vemotherboard.terminal_2**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_R4.vemotherboard.terminal_3**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_R4.vemotherboard.terminal_4**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_R4.vemotherboard.timer_0_1**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_VE_Cortex_R4.vemotherboard.timer_0_1.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R4.vemotherboard.timer_0_1.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_R4.vemotherboard.timer_0_1.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_VE_Cortex_R4.vemotherboard.timer_0_1.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_VE_Cortex_R4.vemotherboard.timer_2_3**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_VE_Cortex_R4.vemotherboard.timer_2_3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R4.vemotherboard.timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R4.vemotherboard.timer_2_3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_VE_Cortex_R4.vemotherboard.timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_VE_Cortex_R4.vemotherboard.ve_sysregs**

Type: `VE_SysRegs`.

**FVP_VE_Cortex_R4.vemotherboard.video_ram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_R4.vemotherboard.vis**

Display window for VE using Visualisation library.
Type: `VEVisualisation`.

**FVP_VE_Cortex_R4.vemotherboard.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_VE_Cortex_R4.vemotherboard.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**`FVP_VE_Cortex_R4.vemotherboard.vis.recorder.playbackTimer`**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**`FVP_VE_Cortex_R4.vemotherboard.vis.recorder.playbackTimer.timer`**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**`FVP_VE_Cortex_R4.vemotherboard.vis.recorder.playbackTimer.timer.thread`**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**`FVP_VE_Cortex_R4.vemotherboard.vis.recorder.playbackTimer.timer.thread_event`**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**`FVP_VE_Cortex_R4.vemotherboard.vis.recorder.recordingDivider`**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

## 15.22 FVP_VE_Cortex-R5x1

List of instances in FVP_VE_Cortex-R5x1.

### FVP_VE_Cortex-R5x1 instances

**`FVP_VE_Cortex_R5x1`**

Top level component of the Cortex_R5x1 Versatile Express inspired model.
Type: `FVP_VE_Cortex_R5x1`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_R5x1.daughterboard**

Cortex-R5x1 DaughterBoard for Versatile Express.
Type: `VEDaughterBoardCortex_R5x1`.

**FVP_VE_Cortex_R5x1.daughterboard.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R5x1.daughterboard.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R5x1.daughterboard.core**

ARM CORTEXR5 Cluster CT model.
Type: `Cluster_ARM_Cortex-R5`.

**FVP_VE_Cortex_R5x1.daughterboard.core.cpu0**

ARM CORTEXR5 CT model.
Type: `ARM_Cortex-R5`.

**FVP_VE_Cortex_R5x1.daughterboard.dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_R5x1.daughterboard.exclusive_monitor**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_VE_Cortex_R5x1.daughterboard.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_VE_Cortex_R5x1.daughterboard.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

**FVP_VE_Cortex_R5x1.daughterboard.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_R5x1.daughterboard.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_R5x1.daughterboard.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_R5x1.daughterboard.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_R5x1.daughterboard.pl310_l2cc**

ARM PrimeCell Level 2 Cache Controller (PL310).
Type: `PL310_L2CC`.

**FVP_VE_Cortex_R5x1.daughterboard.pl390_gic**

Generic Interrupt Controller (PL390).
Type: `PL390_GIC`.

**FVP_VE_Cortex_R5x1.daughterboard.veinterruptmapper**

Interrupt Mapping peripheral (non-cascaded).
Type: `VEInterruptMapper`.

**FVP_VE_Cortex_R5x1.vemotherboard**

Model inspired by the ARM Versatile Express Motherboard for R profile.
Type: `VEMotherBoardR`.

**FVP_VE_Cortex_R5x1.vemotherboard.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**FVP_VE_Cortex_R5x1.vemotherboard.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R5x1.vemotherboard.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_R5x1.vemotherboard.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R5x1.vemotherboard.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R5x1.vemotherboard.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R5x1.vemotherboard.dummy_ram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_R5x1.vemotherboard.dummy_usb**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_R5x1.vemotherboard.flash0**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_VE_Cortex_R5x1.vemotherboard.flash1**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_VE_Cortex_R5x1.vemotherboard.flashloader0**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_VE_Cortex_R5x1.vemotherboard.flashloader1**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_VE_Cortex_R5x1.vemotherboard.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_VE_Cortex_R5x1.vemotherboard.mmc**

Generic Multimedia Card.
Type: `MMC`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_R5x1.vemotherboard.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_R5x1.vemotherboard.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_R5x1.vemotherboard.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_R5x1.vemotherboard.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_R5x1.vemotherboard.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_R5x1.vemotherboard.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R5x1.vemotherboard.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_R5x1.vemotherboard.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_R5x1.vemotherboard.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_R5x1.vemotherboard.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_R5x1.vemotherboard.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_R5x1.vemotherboard.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R5x1.vemotherboard.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_R5x1.vemotherboard.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_R5x1.vemotherboard.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_R5x1.vemotherboard.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_R5x1.vemotherboard.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_R5x1.vemotherboard.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R5x1.vemotherboard.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_R5x1.vemotherboard.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_R5x1.vemotherboard.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_R5x1.vemotherboard.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_R5x1.vemotherboard.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_R5x1.vemotherboard.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R5x1.vemotherboard.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_R5x1.vemotherboard.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_R5x1.vemotherboard.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_R5x1.vemotherboard.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_R5x1.vemotherboard.pl011_uart4**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_R5x1.vemotherboard.pl011_uart4.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R5x1.vemotherboard.pl011_uart4.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ClockTimerThread`.

**FVP_VE_Cortex_R5x1.vemotherboard.pl011_uart4.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_R5x1.vemotherboard.pl011_uart4.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_R5x1.vemotherboard.pl011_uart4.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_R5x1.vemotherboard.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

**FVP_VE_Cortex_R5x1.vemotherboard.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_R5x1.vemotherboard.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_R5x1.vemotherboard.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_R5x1.vemotherboard.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_R5x1.vemotherboard.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_VE_Cortex_R5x1.vemotherboard.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_R5x1.vemotherboard.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_R5x1.vemotherboard.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_R5x1.vemotherboard.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_R5x1.vemotherboard.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_VE_Cortex_R5x1.vemotherboard.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R5x1.vemotherboard.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_VE_Cortex_R5x1.vemotherboard.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R5x1.vemotherboard.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_VE_Cortex_R5x1.vemotherboard.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

**FVP_VE_Cortex_R5x1.vemotherboard.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_R5x1.vemotherboard.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_R5x1.vemotherboard.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_R5x1.vemotherboard.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_R5x1.vemotherboard.pl180_mci**

ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

**FVP_VE_Cortex_R5x1.vemotherboard.ps2keyboard**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_R5x1.vemotherboard.ps2keyboard.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_R5x1.vemotherboard.ps2keyboard.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_R5x1.vemotherboard.ps2keyboard.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_R5x1.vemotherboard.ps2keyboard.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_R5x1.vemotherboard.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

**FVP_VE_Cortex_R5x1.vemotherboard.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_R5x1.vemotherboard.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_R5x1.vemotherboard.ps2mouse.ps2_clocktimer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_R5x1.vemotherboard.ps2mouse.ps2_clocktimer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_R5x1.vemotherboard.psram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_R5x1.vemotherboard.smsc_91c111**
SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_VE_Cortex_R5x1.vemotherboard.sp805_wdog**
ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_VE_Cortex_R5x1.vemotherboard.sp805_wdog.clocktimer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_R5x1.vemotherboard.sp805_wdog.clocktimer.timer**
A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_R5x1.vemotherboard.sp805_wdog.clocktimer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_R5x1.vemotherboard.sp805_wdog.clocktimer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_R5x1.vemotherboard.sp810_sysctrl**

Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_VE_Cortex_R5x1.vemotherboard.sp810_sysctrl.clkdiv_clk0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R5x1.vemotherboard.sp810_sysctrl.clkdiv_clk1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R5x1.vemotherboard.sp810_sysctrl.clkdiv_clk2**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R5x1.vemotherboard.sp810_sysctrl.clkdiv_clk3**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R5x1.vemotherboard.terminal_0**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_R5x1.vemotherboard.terminal_1**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_R5x1.vemotherboard.terminal_2**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_R5x1.vemotherboard.terminal_3**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_R5x1.vemotherboard.terminal_4**

Telnet terminal interface.
Type: `TelnetTerminal`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_R5x1.vemotherboard.timer_0_1**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_VE_Cortex_R5x1.vemotherboard.timer_0_1.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R5x1.vemotherboard.timer_0_1.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R5x1.vemotherboard.timer_0_1.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_VE_Cortex_R5x1.vemotherboard.timer_0_1.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_VE_Cortex_R5x1.vemotherboard.timer_2_3**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_VE_Cortex_R5x1.vemotherboard.timer_2_3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R5x1.vemotherboard.timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R5x1.vemotherboard.timer_2_3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_VE_Cortex_R5x1.vemotherboard.timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_R5x1.vemotherboard.ve_sysregs**

Type: `VE_SysRegs`.

**FVP_VE_Cortex_R5x1.vemotherboard.video_ram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_R5x1.vemotherboard.vis**

Display window for VE using Visualisation library.
Type: `VEVisualisation`.

**FVP_VE_Cortex_R5x1.vemotherboard.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_VE_Cortex_R5x1.vemotherboard.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R5x1.vemotherboard.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_R5x1.vemotherboard.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_R5x1.vemotherboard.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_R5x1.vemotherboard.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_R5x1.vemotherboard.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

# 15.23 FVP_VE_Cortex-R5x2

List of instances in FVP_VE_Cortex-R5x2.

## FVP_VE_Cortex-R5x2 instances

**FVP_VE_Cortex_R5x2**

Top level component of the Cortex_R5x2 Versatile Express inspired model.
Type: `FVP_VE_Cortex_R5x2`.

**FVP_VE_Cortex_R5x2.daughterboard**

Cortex-R5x2 DaughterBoard for Versatile Express.
Type: `VEDaughterBoardCortex_R5x2`.

**FVP_VE_Cortex_R5x2.daughterboard.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R5x2.daughterboard.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R5x2.daughterboard.core**

ARM CORTEXR5 Cluster CT model.
Type: `Cluster_ARM_Cortex-R5`.

**FVP_VE_Cortex_R5x2.daughterboard.core.cpu0**

ARM CORTEXR5 CT model.
Type: `ARM_Cortex-R5`.

**FVP_VE_Cortex_R5x2.daughterboard.core.cpu1**

ARM CORTEXR5 CT model.
Type: `ARM_Cortex-R5`.

**FVP_VE_Cortex_R5x2.daughterboard.dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_R5x2.daughterboard.exclusive_monitor**

Global exclusive monitor.
Type: PVBusExclusiveMonitor.

**FVP_VE_Cortex_R5x2.daughterboard.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: PL111_CLCD.

**FVP_VE_Cortex_R5x2.daughterboard.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: PL11x_CLCD.

**FVP_VE_Cortex_R5x2.daughterboard.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: ClockTimerThread.

**FVP_VE_Cortex_R5x2.daughterboard.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: ClockTimerThread64.

**FVP_VE_Cortex_R5x2.daughterboard.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: SchedulerThread.

**FVP_VE_Cortex_R5x2.daughterboard.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: SchedulerThreadEvent.

**FVP_VE_Cortex_R5x2.daughterboard.pl310_l2cc**

ARM PrimeCell Level 2 Cache Controller (PL310).
Type: PL310_L2CC.

**FVP_VE_Cortex_R5x2.daughterboard.pl390_gic**

Generic Interrupt Controller (PL390).
Type: PL390_GIC.

**FVP_VE_Cortex_R5x2.daughterboard.veinterruptmapper**

Interrupt Mapping peripheral (non-cascaded).
Type: VEInterruptMapper.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_R5x2.vemotherboard**

Model inspired by the ARM Versatile Express Motherboard for R profile.
Type: `VEMotherBoardR`.

**FVP_VE_Cortex_R5x2.vemotherboard.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**FVP_VE_Cortex_R5x2.vemotherboard.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R5x2.vemotherboard.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R5x2.vemotherboard.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R5x2.vemotherboard.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R5x2.vemotherboard.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R5x2.vemotherboard.dummy_ram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_R5x2.vemotherboard.dummy_usb**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_R5x2.vemotherboard.flash0**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_VE_Cortex_R5x2.vemotherboard.flash1**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_VE_Cortex_R5x2.vemotherboard.flashloader0**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_VE_Cortex_R5x2.vemotherboard.flashloader1**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_VE_Cortex_R5x2.vemotherboard.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_VE_Cortex_R5x2.vemotherboard.mmc**

Generic Multimedia Card.
Type: `MMC`.

**FVP_VE_Cortex_R5x2.vemotherboard.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_R5x2.vemotherboard.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_R5x2.vemotherboard.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_R5x2.vemotherboard.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_R5x2.vemotherboard.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_R5x2.vemotherboard.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R5x2.vemotherboard.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_R5x2.vemotherboard.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_R5x2.vemotherboard.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_R5x2.vemotherboard.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_R5x2.vemotherboard.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_R5x2.vemotherboard.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_R5x2.vemotherboard.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_R5x2.vemotherboard.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_R5x2.vemotherboard.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_R5x2.vemotherboard.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_R5x2.vemotherboard.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_R5x2.vemotherboard.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R5x2.vemotherboard.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_R5x2.vemotherboard.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_R5x2.vemotherboard.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_R5x2.vemotherboard.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_R5x2.vemotherboard.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_R5x2.vemotherboard.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R5x2.vemotherboard.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_R5x2.vemotherboard.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_R5x2.vemotherboard.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_R5x2.vemotherboard.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_R5x2.vemotherboard.pl011_uart4**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_R5x2.vemotherboard.pl011_uart4.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R5x2.vemotherboard.pl011_uart4.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_R5x2.vemotherboard.pl011_uart4.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_R5x2.vemotherboard.pl011_uart4.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_R5x2.vemotherboard.pl011_uart4.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_R5x2.vemotherboard.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

**FVP_VE_Cortex_R5x2.vemotherboard.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal()

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_R5x2.vemotherboard.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_R5x2.vemotherboard.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_R5x2.vemotherboard.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_R5x2.vemotherboard.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_VE_Cortex_R5x2.vemotherboard.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_R5x2.vemotherboard.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_R5x2.vemotherboard.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_R5x2.vemotherboard.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_R5x2.vemotherboard.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_VE_Cortex_R5x2.vemotherboard.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R5x2.vemotherboard.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_VE_Cortex_R5x2.vemotherboard.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R5x2.vemotherboard.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_VE_Cortex_R5x2.vemotherboard.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

**FVP_VE_Cortex_R5x2.vemotherboard.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_R5x2.vemotherboard.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_R5x2.vemotherboard.pl111_clcd.pl11x_clcd.timer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_R5x2.vemotherboard.pl111_clcd.pl11x_clcd.timer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_R5x2.vemotherboard.pl180_mci**
ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

**FVP_VE_Cortex_R5x2.vemotherboard.ps2keyboard**
Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

**FVP_VE_Cortex_R5x2.vemotherboard.ps2keyboard.ps2_clocktimer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_R5x2.vemotherboard.ps2keyboard.ps2_clocktimer.timer**
A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_R5x2.vemotherboard.ps2keyboard.ps2_clocktimer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_R5x2.vemotherboard.ps2keyboard.ps2_clocktimer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_R5x2.vemotherboard.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

**FVP_VE_Cortex_R5x2.vemotherboard.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_R5x2.vemotherboard.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_R5x2.vemotherboard.ps2mouse.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_R5x2.vemotherboard.ps2mouse.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_R5x2.vemotherboard.psram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_R5x2.vemotherboard.smsc_91c111**

SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_VE_Cortex_R5x2.vemotherboard.sp805_wdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_VE_Cortex_R5x2.vemotherboard.sp805_wdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_R5x2.vemotherboard.sp805_wdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_R5x2.vemotherboard.sp805_wdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_R5x2.vemotherboard.sp805_wdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_R5x2.vemotherboard.sp810_sysctrl**

Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_VE_Cortex_R5x2.vemotherboard.sp810_sysctrl.clkdiv_clk0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R5x2.vemotherboard.sp810_sysctrl.clkdiv_clk1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R5x2.vemotherboard.sp810_sysctrl.clkdiv_clk2**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R5x2.vemotherboard.sp810_sysctrl.clkdiv_clk3**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R5x2.vemotherboard.terminal_0**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_R5x2.vemotherboard.terminal_1**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_R5x2.vemotherboard.terminal_2**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_R5x2.vemotherboard.terminal_3**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_R5x2.vemotherboard.terminal_4**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_R5x2.vemotherboard.timer_0_1**
ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_VE_Cortex_R5x2.vemotherboard.timer_0_1.clk_div0**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R5x2.vemotherboard.timer_0_1.clk_div1**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R5x2.vemotherboard.timer_0_1.counter0**
Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_VE_Cortex_R5x2.vemotherboard.timer_0_1.counter1**
Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_VE_Cortex_R5x2.vemotherboard.timer_2_3**
ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_R5x2.vemotherboard.timer_2_3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R5x2.vemotherboard.timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R5x2.vemotherboard.timer_2_3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_VE_Cortex_R5x2.vemotherboard.timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_VE_Cortex_R5x2.vemotherboard.ve_sysregs**

Type: `VE_SysRegs`.

**FVP_VE_Cortex_R5x2.vemotherboard.video_ram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_R5x2.vemotherboard.vis**

Display window for VE using Visualisation library.
Type: `VEVisualisation`.

**FVP_VE_Cortex_R5x2.vemotherboard.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_VE_Cortex_R5x2.vemotherboard.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R5x2.vemotherboard.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**`FVP_VE_Cortex_R5x2.vemotherboard.vis.recorder.playbackTimer.timer`**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**`FVP_VE_Cortex_R5x2.vemotherboard.vis.recorder.playbackTimer.timer.thread`**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**`FVP_VE_Cortex_R5x2.vemotherboard.vis.recorder.playbackTimer.timer.thread_event`**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**`FVP_VE_Cortex_R5x2.vemotherboard.vis.recorder.recordingDivider`**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

## 15.24 FVP_VE_Cortex-R7x1

List of instances in FVP_VE_Cortex-R7x1.

### FVP_VE_Cortex-R7x1 instances

**`FVP_VE_Cortex_R7x1`**

Top level component of the Cortex_R7x1 Versatile Express inspired model.
Type: `FVP_VE_Cortex_R7x1`.

**`FVP_VE_Cortex_R7x1.clockdivider`**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**`FVP_VE_Cortex_R7x1.daughterboard`**

Cortex_R7x1 DaughterBoard for Versatile Express.
Type: `VEDaughterBoardCortex_R7x1`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_R7x1.daughterboard.core**
ARM CORTEXR7 Cluster CT model.
Type: `Cluster_ARM_Cortex-R7`.

**FVP_VE_Cortex_R7x1.daughterboard.core.cpu0**
ARM CORTEXR7 CT model.
Type: `ARM_Cortex-R7`.

**FVP_VE_Cortex_R7x1.daughterboard.core.cpu0.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_R7x1.daughterboard.core.cpu0.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_R7x1.daughterboard.dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_R7x1.daughterboard.exclusive_monitor**
Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_VE_Cortex_R7x1.daughterboard.periph_clockdivider**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R7x1.daughterboard.pl310_l2cc**
ARM PrimeCell Level 2 Cache Controller (PL310).
Type: `PL310_L2CC`.

**FVP_VE_Cortex_R7x1.daughterboard.veinterruptmapper**
Interrupt Mapping peripheral (non-cascaded).
Type: `VEInterruptMapper`.

**FVP_VE_Cortex_R7x1.vemotherboard**
Model inspired by the ARM Versatile Express Motherboard for R profile.
Type: `VEMotherBoardR`.

**FVP_VE_Cortex_R7x1.vemotherboard.audioout**
SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**FVP_VE_Cortex_R7x1.vemotherboard.clock100Hz**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_R7x1.vemotherboard.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R7x1.vemotherboard.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R7x1.vemotherboard.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R7x1.vemotherboard.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R7x1.vemotherboard.dummy_ram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_R7x1.vemotherboard.dummy_usb**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_R7x1.vemotherboard.flash0**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_VE_Cortex_R7x1.vemotherboard.flash1**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_VE_Cortex_R7x1.vemotherboard.flashloader0**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_VE_Cortex_R7x1.vemotherboard.flashloader1**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_R7x1.vemotherboard.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_VE_Cortex_R7x1.vemotherboard.mmc**

Generic Multimedia Card.
Type: `MMC`.

**FVP_VE_Cortex_R7x1.vemotherboard.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_R7x1.vemotherboard.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_R7x1.vemotherboard.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_R7x1.vemotherboard.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_R7x1.vemotherboard.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_R7x1.vemotherboard.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R7x1.vemotherboard.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_R7x1.vemotherboard.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_R7x1.vemotherboard.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_R7x1.vemotherboard.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_R7x1.vemotherboard.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_R7x1.vemotherboard.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R7x1.vemotherboard.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_R7x1.vemotherboard.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_R7x1.vemotherboard.pl011_uart1.timer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_R7x1.vemotherboard.pl011_uart1.timer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_R7x1.vemotherboard.pl011_uart2**
ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_R7x1.vemotherboard.pl011_uart2.clk_divider**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R7x1.vemotherboard.pl011_uart2.timer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_R7x1.vemotherboard.pl011_uart2.timer.timer**
A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_R7x1.vemotherboard.pl011_uart2.timer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_R7x1.vemotherboard.pl011_uart2.timer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_R7x1.vemotherboard.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_R7x1.vemotherboard.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R7x1.vemotherboard.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_R7x1.vemotherboard.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_R7x1.vemotherboard.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_R7x1.vemotherboard.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_R7x1.vemotherboard.pl011_uart4**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_R7x1.vemotherboard.pl011_uart4.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_R7x1.vemotherboard.pl011_uart4.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_R7x1.vemotherboard.pl011_uart4.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_R7x1.vemotherboard.pl011_uart4.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_R7x1.vemotherboard.pl011_uart4.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_R7x1.vemotherboard.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

**FVP_VE_Cortex_R7x1.vemotherboard.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_R7x1.vemotherboard.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_R7x1.vemotherboard.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_R7x1.vemotherboard.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_R7x1.vemotherboard.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_VE_Cortex_R7x1.vemotherboard.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_R7x1.vemotherboard.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_R7x1.vemotherboard.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_R7x1.vemotherboard.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_R7x1.vemotherboard.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_VE_Cortex_R7x1.vemotherboard.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_R7x1.vemotherboard.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_VE_Cortex_R7x1.vemotherboard.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R7x1.vemotherboard.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_VE_Cortex_R7x1.vemotherboard.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

**FVP_VE_Cortex_R7x1.vemotherboard.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_R7x1.vemotherboard.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_R7x1.vemotherboard.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_R7x1.vemotherboard.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_R7x1.vemotherboard.pl180_mci**

ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_R7x1.vemotherboard.ps2keyboard**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

**FVP_VE_Cortex_R7x1.vemotherboard.ps2keyboard.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_R7x1.vemotherboard.ps2keyboard.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_R7x1.vemotherboard.ps2keyboard.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_R7x1.vemotherboard.ps2keyboard.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_R7x1.vemotherboard.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

**FVP_VE_Cortex_R7x1.vemotherboard.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_R7x1.vemotherboard.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_R7x1.vemotherboard.ps2mouse.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_R7x1.vemotherboard.ps2mouse.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_R7x1.vemotherboard.psram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_R7x1.vemotherboard.smsc_91c111**

SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_VE_Cortex_R7x1.vemotherboard.sp805_wdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_VE_Cortex_R7x1.vemotherboard.sp805_wdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_R7x1.vemotherboard.sp805_wdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_R7x1.vemotherboard.sp805_wdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `SchedulerThread`.

**FVP_VE_Cortex_R7x1.vemotherboard.sp805_wdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_R7x1.vemotherboard.sp810_sysctrl**

Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_VE_Cortex_R7x1.vemotherboard.sp810_sysctrl.clkdiv_clk0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R7x1.vemotherboard.sp810_sysctrl.clkdiv_clk1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R7x1.vemotherboard.sp810_sysctrl.clkdiv_clk2**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R7x1.vemotherboard.sp810_sysctrl.clkdiv_clk3**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R7x1.vemotherboard.terminal_0**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_R7x1.vemotherboard.terminal_1**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_R7x1.vemotherboard.terminal_2**

Telnet terminal interface.
Type: `TelnetTerminal`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_R7x1.vemotherboard.terminal_3**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_R7x1.vemotherboard.terminal_4**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_R7x1.vemotherboard.timer_0_1**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_VE_Cortex_R7x1.vemotherboard.timer_0_1.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R7x1.vemotherboard.timer_0_1.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R7x1.vemotherboard.timer_0_1.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_VE_Cortex_R7x1.vemotherboard.timer_0_1.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_VE_Cortex_R7x1.vemotherboard.timer_2_3**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_VE_Cortex_R7x1.vemotherboard.timer_2_3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R7x1.vemotherboard.timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_R7x1.vemotherboard.timer_2_3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_VE_Cortex_R7x1.vemotherboard.timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_VE_Cortex_R7x1.vemotherboard.ve_sysregs**

Type: `VE_SysRegs`.

**FVP_VE_Cortex_R7x1.vemotherboard.video_ram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_R7x1.vemotherboard.vis**

Display window for VE using Visualisation library.
Type: `VEVisualisation`.

**FVP_VE_Cortex_R7x1.vemotherboard.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_VE_Cortex_R7x1.vemotherboard.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R7x1.vemotherboard.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_R7x1.vemotherboard.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_R7x1.vemotherboard.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `SchedulerThread`.

**`FVP_VE_Cortex_R7x1.vemotherboard.vis.recorder.playbackTimer.timer.thread_event`**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**`FVP_VE_Cortex_R7x1.vemotherboard.vis.recorder.recordingDivider`**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

## 15.25 FVP_VE_Cortex-R7x2

List of instances in FVP_VE_Cortex-R7x2.

### FVP_VE_Cortex-R7x2 instances

**`FVP_VE_Cortex_R7x2`**

Top level component of the Cortex_R7x2 Versatile Express inspired model.
Type: `FVP_VE_Cortex_R7x2`.

**`FVP_VE_Cortex_R7x2.clockdivider`**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**`FVP_VE_Cortex_R7x2.daughterboard`**

Cortex_R7x2 DaughterBoard for Versatile Express.
Type: `VEDaughterBoardCortex_R7x2`.

**`FVP_VE_Cortex_R7x2.daughterboard.core`**

ARM CORTEXR7 Cluster CT model.
Type: `Cluster_ARM_Cortex-R7`.

**`FVP_VE_Cortex_R7x2.daughterboard.core.cpu0`**

ARM CORTEXR7 CT model.
Type: `ARM_Cortex-R7`.

**`FVP_VE_Cortex_R7x2.daughterboard.core.cpu0.l1dcache`**

PV Cache.
Type: `PVCache`.

**`FVP_VE_Cortex_R7x2.daughterboard.core.cpu0.l1icache`**

PV Cache.
Type: `PVCache`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_R7x2.daughterboard.core.cpu1**

ARM CORTEXR7 CT model.
Type: `ARM_Cortex-R7`.

**FVP_VE_Cortex_R7x2.daughterboard.core.cpu1.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_R7x2.daughterboard.core.cpu1.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_R7x2.daughterboard.dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_R7x2.daughterboard.exclusive_monitor**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_VE_Cortex_R7x2.daughterboard.periph_clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R7x2.daughterboard.pl310_l2cc**

ARM PrimeCell Level 2 Cache Controller (PL310).
Type: `PL310_L2CC`.

**FVP_VE_Cortex_R7x2.daughterboard.veinterruptmapper**

Interrupt Mapping peripheral (non-cascaded).
Type: `VEInterruptMapper`.

**FVP_VE_Cortex_R7x2.vemotherboard**

Model inspired by the ARM Versatile Express Motherboard for R profile.
Type: `VEMotherBoardR`.

**FVP_VE_Cortex_R7x2.vemotherboard.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**FVP_VE_Cortex_R7x2.vemotherboard.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R7x2.vemotherboard.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R7x2.vemotherboard.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R7x2.vemotherboard.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R7x2.vemotherboard.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R7x2.vemotherboard.dummy_ram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_R7x2.vemotherboard.dummy_usb**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_R7x2.vemotherboard.flash0**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_VE_Cortex_R7x2.vemotherboard.flash1**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_VE_Cortex_R7x2.vemotherboard.flashloader0**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_VE_Cortex_R7x2.vemotherboard.flashloader1**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_VE_Cortex_R7x2.vemotherboard.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_R7x2.vemotherboard.mmc**

Generic Multimedia Card.
Type: `MMC`.

**FVP_VE_Cortex_R7x2.vemotherboard.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_R7x2.vemotherboard.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_R7x2.vemotherboard.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_R7x2.vemotherboard.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_R7x2.vemotherboard.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_R7x2.vemotherboard.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R7x2.vemotherboard.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ClockTimerThread`.

**FVP_VE_Cortex_R7x2.vemotherboard.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_R7x2.vemotherboard.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_R7x2.vemotherboard.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_R7x2.vemotherboard.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_R7x2.vemotherboard.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R7x2.vemotherboard.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_R7x2.vemotherboard.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_R7x2.vemotherboard.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_R7x2.vemotherboard.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_R7x2.vemotherboard.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_R7x2.vemotherboard.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R7x2.vemotherboard.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_R7x2.vemotherboard.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_R7x2.vemotherboard.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_R7x2.vemotherboard.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_R7x2.vemotherboard.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_R7x2.vemotherboard.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R7x2.vemotherboard.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_R7x2.vemotherboard.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_R7x2.vemotherboard.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_R7x2.vemotherboard.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_R7x2.vemotherboard.pl011_uart4**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_R7x2.vemotherboard.pl011_uart4.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R7x2.vemotherboard.pl011_uart4.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_R7x2.vemotherboard.pl011_uart4.timer.timer**
A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_R7x2.vemotherboard.pl011_uart4.timer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_R7x2.vemotherboard.pl011_uart4.timer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_R7x2.vemotherboard.pl031_rtc**
ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

**FVP_VE_Cortex_R7x2.vemotherboard.pl031_rtc.timer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_R7x2.vemotherboard.pl031_rtc.timer.timer**
A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_R7x2.vemotherboard.pl031_rtc.timer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_R7x2.vemotherboard.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_R7x2.vemotherboard.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_VE_Cortex_R7x2.vemotherboard.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_R7x2.vemotherboard.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_R7x2.vemotherboard.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_R7x2.vemotherboard.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_R7x2.vemotherboard.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_VE_Cortex_R7x2.vemotherboard.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R7x2.vemotherboard.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_R7x2.vemotherboard.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R7x2.vemotherboard.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_VE_Cortex_R7x2.vemotherboard.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

**FVP_VE_Cortex_R7x2.vemotherboard.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_R7x2.vemotherboard.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_R7x2.vemotherboard.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_R7x2.vemotherboard.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_R7x2.vemotherboard.pl180_mci**

ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

**FVP_VE_Cortex_R7x2.vemotherboard.ps2keyboard**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `PS2Keyboard`.

**FVP_VE_Cortex_R7x2.vemotherboard.ps2keyboard.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_R7x2.vemotherboard.ps2keyboard.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_R7x2.vemotherboard.ps2keyboard.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_R7x2.vemotherboard.ps2keyboard.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_R7x2.vemotherboard.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

**FVP_VE_Cortex_R7x2.vemotherboard.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_R7x2.vemotherboard.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_R7x2.vemotherboard.ps2mouse.ps2_clocktimer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_R7x2.vemotherboard.ps2mouse.ps2_clocktimer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_R7x2.vemotherboard.psram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_R7x2.vemotherboard.smsc_91c111**
SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_VE_Cortex_R7x2.vemotherboard.sp805_wdog**
ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_VE_Cortex_R7x2.vemotherboard.sp805_wdog.clocktimer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_R7x2.vemotherboard.sp805_wdog.clocktimer.timer**
A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_R7x2.vemotherboard.sp805_wdog.clocktimer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_R7x2.vemotherboard.sp805_wdog.clocktimer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_R7x2.vemotherboard.sp810_sysctrl**

Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_VE_Cortex_R7x2.vemotherboard.sp810_sysctrl.clkdiv_clk0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R7x2.vemotherboard.sp810_sysctrl.clkdiv_clk1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R7x2.vemotherboard.sp810_sysctrl.clkdiv_clk2**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R7x2.vemotherboard.sp810_sysctrl.clkdiv_clk3**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R7x2.vemotherboard.terminal_0**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_R7x2.vemotherboard.terminal_1**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_R7x2.vemotherboard.terminal_2**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_R7x2.vemotherboard.terminal_3**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_R7x2.vemotherboard.terminal_4**

Telnet terminal interface.
Type: `TelnetTerminal`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_R7x2.vemotherboard.timer_0_1**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_VE_Cortex_R7x2.vemotherboard.timer_0_1.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R7x2.vemotherboard.timer_0_1.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R7x2.vemotherboard.timer_0_1.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_VE_Cortex_R7x2.vemotherboard.timer_0_1.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_VE_Cortex_R7x2.vemotherboard.timer_2_3**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_VE_Cortex_R7x2.vemotherboard.timer_2_3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R7x2.vemotherboard.timer_2_3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R7x2.vemotherboard.timer_2_3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_VE_Cortex_R7x2.vemotherboard.timer_2_3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_R7x2.vemotherboard.ve_sysregs**

Type: `VE_SysRegs`.

**FVP_VE_Cortex_R7x2.vemotherboard.video_ram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_R7x2.vemotherboard.vis**

Display window for VE using Visualisation library.
Type: `VEVisualisation`.

**FVP_VE_Cortex_R7x2.vemotherboard.vis.recorder**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**FVP_VE_Cortex_R7x2.vemotherboard.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R7x2.vemotherboard.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_R7x2.vemotherboard.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_R7x2.vemotherboard.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_R7x2.vemotherboard.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_R7x2.vemotherboard.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

## 15.26 FVP_VE_Cortex-R8x1

List of instances in FVP_VE_Cortex-R8x1.

### FVP_VE_Cortex-R8x1 instances

**FVP_VE_Cortex_R8x1**

Top level component of the Cortex_R8x1 Versatile Express inspired model.
Type: `FVP_VE_Cortex_R8x1`.

**FVP_VE_Cortex_R8x1.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R8x1.daughterboard**

Cortex_R8x1 DaughterBoard for Versatile Express.
Type: `VEDaughterBoardCortex_R8x1`.

**FVP_VE_Cortex_R8x1.daughterboard.core**

ARM CORTEXR8 Cluster CT model.
Type: `Cluster_ARM_Cortex-R8`.

**FVP_VE_Cortex_R8x1.daughterboard.core.cpu0**

ARM CORTEXR8 CT model.
Type: `ARM_Cortex-R8`.

**FVP_VE_Cortex_R8x1.daughterboard.core.cpu0.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_R8x1.daughterboard.core.cpu0.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_R8x1.daughterboard.dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_R8x1.daughterboard.exclusive_monitor**

Global exclusive monitor.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `PVBusExclusiveMonitor`.

**FVP_VE_Cortex_R8x1.daughterboard.periph_clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R8x1.daughterboard.pl310_l2cc**

ARM PrimeCell Level 2 Cache Controller (PL310).
Type: `PL310_L2CC`.

**FVP_VE_Cortex_R8x1.daughterboard.veinterruptmapper**

Interrupt Mapping peripheral (non-cascaded).
Type: `VEInterruptMapper`.

**FVP_VE_Cortex_R8x1.vemotherboard**

Model inspired by the ARM Versatile Express Motherboard for R profile.
Type: `VEMotherBoardR`.

**FVP_VE_Cortex_R8x1.vemotherboard.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**FVP_VE_Cortex_R8x1.vemotherboard.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R8x1.vemotherboard.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R8x1.vemotherboard.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R8x1.vemotherboard.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_R8x1.vemotherboard.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R8x1.vemotherboard.dummy_ram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_R8x1.vemotherboard.dummy_usb**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_R8x1.vemotherboard.flash0**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_VE_Cortex_R8x1.vemotherboard.flash1**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_VE_Cortex_R8x1.vemotherboard.flashloader0**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_VE_Cortex_R8x1.vemotherboard.flashloader1**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_VE_Cortex_R8x1.vemotherboard.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_VE_Cortex_R8x1.vemotherboard.mmc**

Generic Multimedia Card.
Type: `MMC`.

**FVP_VE_Cortex_R8x1.vemotherboard.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_R8x1.vemotherboard.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_R8x1.vemotherboard.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_R8x1.vemotherboard.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_R8x1.vemotherboard.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_R8x1.vemotherboard.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R8x1.vemotherboard.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_R8x1.vemotherboard.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_R8x1.vemotherboard.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_R8x1.vemotherboard.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_R8x1.vemotherboard.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_R8x1.vemotherboard.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R8x1.vemotherboard.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_R8x1.vemotherboard.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_R8x1.vemotherboard.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_R8x1.vemotherboard.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_R8x1.vemotherboard.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_R8x1.vemotherboard.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R8x1.vemotherboard.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_R8x1.vemotherboard.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_R8x1.vemotherboard.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_R8x1.vemotherboard.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_R8x1.vemotherboard.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_R8x1.vemotherboard.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R8x1.vemotherboard.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ClockTimerThread`.

**FVP_VE_Cortex_R8x1.vemotherboard.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_R8x1.vemotherboard.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_R8x1.vemotherboard.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_R8x1.vemotherboard.pl011_uart4**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_R8x1.vemotherboard.pl011_uart4.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R8x1.vemotherboard.pl011_uart4.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_R8x1.vemotherboard.pl011_uart4.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_R8x1.vemotherboard.pl011_uart4.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_R8x1.vemotherboard.pl011_uart4.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_R8x1.vemotherboard.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

**FVP_VE_Cortex_R8x1.vemotherboard.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_R8x1.vemotherboard.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_R8x1.vemotherboard.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_R8x1.vemotherboard.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_R8x1.vemotherboard.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_VE_Cortex_R8x1.vemotherboard.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_R8x1.vemotherboard.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_R8x1.vemotherboard.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_R8x1.vemotherboard.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_R8x1.vemotherboard.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_VE_Cortex_R8x1.vemotherboard.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R8x1.vemotherboard.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_VE_Cortex_R8x1.vemotherboard.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R8x1.vemotherboard.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_VE_Cortex_R8x1.vemotherboard.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_R8x1.vemotherboard.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_R8x1.vemotherboard.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_R8x1.vemotherboard.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_R8x1.vemotherboard.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_R8x1.vemotherboard.pl180_mci**

ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

**FVP_VE_Cortex_R8x1.vemotherboard.ps2keyboard**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

**FVP_VE_Cortex_R8x1.vemotherboard.ps2keyboard.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_R8x1.vemotherboard.ps2keyboard.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_R8x1.vemotherboard.ps2keyboard.ps2_clocktimer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_R8x1.vemotherboard.ps2keyboard.ps2_clocktimer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_R8x1.vemotherboard.ps2mouse**
Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

**FVP_VE_Cortex_R8x1.vemotherboard.ps2mouse.ps2_clocktimer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_R8x1.vemotherboard.ps2mouse.ps2_clocktimer.timer**
A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_R8x1.vemotherboard.ps2mouse.ps2_clocktimer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_R8x1.vemotherboard.ps2mouse.ps2_clocktimer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_R8x1.vemotherboard.psram**
RAM device, can be dynamic or static ram.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `RAMDevice`.

**FVP_VE_Cortex_R8x1.vemotherboard.smsc_91c111**

SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_VE_Cortex_R8x1.vemotherboard.sp805_wdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_VE_Cortex_R8x1.vemotherboard.sp805_wdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_R8x1.vemotherboard.sp805_wdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_R8x1.vemotherboard.sp805_wdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_R8x1.vemotherboard.sp805_wdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_R8x1.vemotherboard.sp810_sysctrl**

Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_VE_Cortex_R8x1.vemotherboard.sp810_sysctrl.clkdiv_clk0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R8x1.vemotherboard.sp810_sysctrl.clkdiv_clk1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R8x1.vemotherboard.sp810_sysctrl.clkdiv_clk2**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R8x1.vemotherboard.sp810_sysctrl.clkdiv_clk3**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R8x1.vemotherboard.terminal_0**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_R8x1.vemotherboard.terminal_1**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_R8x1.vemotherboard.terminal_2**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_R8x1.vemotherboard.terminal_3**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_R8x1.vemotherboard.terminal_4**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_R8x1.vemotherboard.timer_0_1**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_VE_Cortex_R8x1.vemotherboard.timer_0_1.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R8x1.vemotherboard.timer_0_1.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R8x1.vemotherboard.timer_0_1.counter0**
Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_VE_Cortex_R8x1.vemotherboard.timer_0_1.counter1**
Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_VE_Cortex_R8x1.vemotherboard.timer_2_3**
ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_VE_Cortex_R8x1.vemotherboard.timer_2_3.clk_div0**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R8x1.vemotherboard.timer_2_3.clk_div1**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R8x1.vemotherboard.timer_2_3.counter0**
Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_VE_Cortex_R8x1.vemotherboard.timer_2_3.counter1**
Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_VE_Cortex_R8x1.vemotherboard.ve_sysregs**
Type: `VE_SysRegs`.

**FVP_VE_Cortex_R8x1.vemotherboard.video_ram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_R8x1.vemotherboard.vis**
Display window for VE using Visualisation library.
Type: `VEVisualisation`.

**FVP_VE_Cortex_R8x1.vemotherboard.vis.recorder**
Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_R8x1.vemotherboard.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R8x1.vemotherboard.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_R8x1.vemotherboard.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_R8x1.vemotherboard.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_R8x1.vemotherboard.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_R8x1.vemotherboard.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

# 15.27 FVP_VE_Cortex-R8x2

List of instances in FVP_VE_Cortex-R8x2.

## FVP_VE_Cortex-R8x2 instances

**FVP_VE_Cortex_R8x2**

Top level component of the Cortex_R8x2 Versatile Express inspired model.
Type: `FVP_VE_Cortex_R8x2`.

**FVP_VE_Cortex_R8x2.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R8x2.daughterboard**

Cortex_R8x2 DaughterBoard for Versatile Express.
Type: `VEDaughterBoardCortex_R8x2`.

**FVP_VE_Cortex_R8x2.daughterboard.core**

ARM CORTEXR8 Cluster CT model.
Type: `Cluster_ARM_Cortex-R8`.

**FVP_VE_Cortex_R8x2.daughterboard.core.cpu0**

ARM CORTEXR8 CT model.
Type: `ARM_Cortex-R8`.

**FVP_VE_Cortex_R8x2.daughterboard.core.cpu0.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_R8x2.daughterboard.core.cpu0.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_R8x2.daughterboard.core.cpu1**

ARM CORTEXR8 CT model.
Type: `ARM_Cortex-R8`.

**FVP_VE_Cortex_R8x2.daughterboard.core.cpu1.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_R8x2.daughterboard.core.cpu1.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_R8x2.daughterboard.dram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_R8x2.daughterboard.exclusive_monitor**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_VE_Cortex_R8x2.daughterboard.periph_clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R8x2.daughterboard.pl310_l2cc**

ARM PrimeCell Level 2 Cache Controller (PL310).
Type: `PL310_L2CC`.

**FVP_VE_Cortex_R8x2.daughterboard.veinterruptmapper**

Interrupt Mapping peripheral (non-cascaded).
Type: `VEInterruptMapper`.

**FVP_VE_Cortex_R8x2.vemotherboard**

Model inspired by the ARM Versatile Express Motherboard for R profile.
Type: `VEMotherBoardR`.

**FVP_VE_Cortex_R8x2.vemotherboard.audioout**

SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

**FVP_VE_Cortex_R8x2.vemotherboard.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R8x2.vemotherboard.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R8x2.vemotherboard.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R8x2.vemotherboard.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R8x2.vemotherboard.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R8x2.vemotherboard.dummy_ram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_R8x2.vemotherboard.dummy_usb**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_R8x2.vemotherboard.flash0**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_VE_Cortex_R8x2.vemotherboard.flash1**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_VE_Cortex_R8x2.vemotherboard.flashloader0**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_VE_Cortex_R8x2.vemotherboard.flashloader1**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_VE_Cortex_R8x2.vemotherboard.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_VE_Cortex_R8x2.vemotherboard.mmc**

Generic Multimedia Card.
Type: `MMC`.

**FVP_VE_Cortex_R8x2.vemotherboard.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_R8x2.vemotherboard.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_R8x2.vemotherboard.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_R8x2.vemotherboard.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_R8x2.vemotherboard.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_R8x2.vemotherboard.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R8x2.vemotherboard.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_R8x2.vemotherboard.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_R8x2.vemotherboard.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_R8x2.vemotherboard.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_R8x2.vemotherboard.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_R8x2.vemotherboard.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R8x2.vemotherboard.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_R8x2.vemotherboard.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_R8x2.vemotherboard.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_R8x2.vemotherboard.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_R8x2.vemotherboard.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_R8x2.vemotherboard.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R8x2.vemotherboard.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_R8x2.vemotherboard.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_R8x2.vemotherboard.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_R8x2.vemotherboard.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_R8x2.vemotherboard.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_R8x2.vemotherboard.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R8x2.vemotherboard.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ClockTimerThread`.

**FVP_VE_Cortex_R8x2.vemotherboard.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_R8x2.vemotherboard.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_R8x2.vemotherboard.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_R8x2.vemotherboard.pl011_uart4**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_R8x2.vemotherboard.pl011_uart4.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R8x2.vemotherboard.pl011_uart4.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_R8x2.vemotherboard.pl011_uart4.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_R8x2.vemotherboard.pl011_uart4.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_R8x2.vemotherboard.pl011_uart4.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_R8x2.vemotherboard.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

**FVP_VE_Cortex_R8x2.vemotherboard.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_R8x2.vemotherboard.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_R8x2.vemotherboard.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_R8x2.vemotherboard.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_R8x2.vemotherboard.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_VE_Cortex_R8x2.vemotherboard.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_R8x2.vemotherboard.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_R8x2.vemotherboard.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_R8x2.vemotherboard.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_R8x2.vemotherboard.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_VE_Cortex_R8x2.vemotherboard.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R8x2.vemotherboard.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_VE_Cortex_R8x2.vemotherboard.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R8x2.vemotherboard.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_VE_Cortex_R8x2.vemotherboard.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_R8x2.vemotherboard.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_R8x2.vemotherboard.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_R8x2.vemotherboard.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_R8x2.vemotherboard.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_R8x2.vemotherboard.pl180_mci**

ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

**FVP_VE_Cortex_R8x2.vemotherboard.ps2keyboard**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

**FVP_VE_Cortex_R8x2.vemotherboard.ps2keyboard.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_R8x2.vemotherboard.ps2keyboard.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_R8x2.vemotherboard.ps2keyboard.ps2_clocktimer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_R8x2.vemotherboard.ps2keyboard.ps2_clocktimer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_R8x2.vemotherboard.ps2mouse**
Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

**FVP_VE_Cortex_R8x2.vemotherboard.ps2mouse.ps2_clocktimer**
A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_R8x2.vemotherboard.ps2mouse.ps2_clocktimer.timer**
A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_R8x2.vemotherboard.ps2mouse.ps2_clocktimer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_R8x2.vemotherboard.ps2mouse.ps2_clocktimer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_R8x2.vemotherboard.psram**
RAM device, can be dynamic or static ram.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `RAMDevice`.

**FVP_VE_Cortex_R8x2.vemotherboard.smsc_91c111**

SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_VE_Cortex_R8x2.vemotherboard.sp805_wdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_VE_Cortex_R8x2.vemotherboard.sp805_wdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_R8x2.vemotherboard.sp805_wdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_R8x2.vemotherboard.sp805_wdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_R8x2.vemotherboard.sp805_wdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_R8x2.vemotherboard.sp810_sysctrl**

Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_VE_Cortex_R8x2.vemotherboard.sp810_sysctrl.clkdiv_clk0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R8x2.vemotherboard.sp810_sysctrl.clkdiv_clk1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R8x2.vemotherboard.sp810_sysctrl.clkdiv_clk2**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R8x2.vemotherboard.sp810_sysctrl.clkdiv_clk3**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R8x2.vemotherboard.terminal_0**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_R8x2.vemotherboard.terminal_1**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_R8x2.vemotherboard.terminal_2**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_R8x2.vemotherboard.terminal_3**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_R8x2.vemotherboard.terminal_4**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_R8x2.vemotherboard.timer_0_1**

ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_VE_Cortex_R8x2.vemotherboard.timer_0_1.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R8x2.vemotherboard.timer_0_1.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**`FVP_VE_Cortex_R8x2.vemotherboard.timer_0_1.counter0`**
Internal component used by SP804 Timer module.
Type: `CounterModule`.

**`FVP_VE_Cortex_R8x2.vemotherboard.timer_0_1.counter1`**
Internal component used by SP804 Timer module.
Type: `CounterModule`.

**`FVP_VE_Cortex_R8x2.vemotherboard.timer_2_3`**
ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**`FVP_VE_Cortex_R8x2.vemotherboard.timer_2_3.clk_div0`**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**`FVP_VE_Cortex_R8x2.vemotherboard.timer_2_3.clk_div1`**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**`FVP_VE_Cortex_R8x2.vemotherboard.timer_2_3.counter0`**
Internal component used by SP804 Timer module.
Type: `CounterModule`.

**`FVP_VE_Cortex_R8x2.vemotherboard.timer_2_3.counter1`**
Internal component used by SP804 Timer module.
Type: `CounterModule`.

**`FVP_VE_Cortex_R8x2.vemotherboard.ve_sysregs`**
Type: `VE_SysRegs`.

**`FVP_VE_Cortex_R8x2.vemotherboard.video_ram`**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**`FVP_VE_Cortex_R8x2.vemotherboard.vis`**
Display window for VE using Visualisation library.
Type: `VEVisualisation`.

**`FVP_VE_Cortex_R8x2.vemotherboard.vis.recorder`**
Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_R8x2.vemotherboard.vis.recorder.playbackDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R8x2.vemotherboard.vis.recorder.playbackTimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_R8x2.vemotherboard.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_R8x2.vemotherboard.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_R8x2.vemotherboard.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_R8x2.vemotherboard.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

# 15.28 FVP_VE_Cortex-R8x4

List of instances in FVP_VE_Cortex-R8x4.

## FVP_VE_Cortex-R8x4 instances

**FVP_VE_Cortex_R8x4**

Top level component of the Cortex_R8x4 Versatile Express inspired model.
Type: `FVP_VE_Cortex_R8x4`.

**FVP_VE_Cortex_R8x4.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R8x4.daughterboard**

Cortex_R8x4 DaughterBoard for Versatile Express.
Type: `VEDaughterBoardCortex_R8x4`.

**FVP_VE_Cortex_R8x4.daughterboard.core**

ARM CORTEXR8 Cluster CT model.
Type: `Cluster_ARM_Cortex-R8`.

**FVP_VE_Cortex_R8x4.daughterboard.core.cpu0**

ARM CORTEXR8 CT model.
Type: `ARM_Cortex-R8`.

**FVP_VE_Cortex_R8x4.daughterboard.core.cpu0.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_R8x4.daughterboard.core.cpu0.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_R8x4.daughterboard.core.cpu1**

ARM CORTEXR8 CT model.
Type: `ARM_Cortex-R8`.

**FVP_VE_Cortex_R8x4.daughterboard.core.cpu1.l1dcache**

PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_R8x4.daughterboard.core.cpu1.l1icache**

PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_R8x4.daughterboard.core.cpu2**

ARM CORTEXR8 CT model.
Type: `ARM_Cortex-R8`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_R8x4.daughterboard.core.cpu2.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_R8x4.daughterboard.core.cpu2.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_R8x4.daughterboard.core.cpu3**
ARM CORTEXR8 CT model.
Type: `ARM_Cortex-R8`.

**FVP_VE_Cortex_R8x4.daughterboard.core.cpu3.l1dcache**
PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_R8x4.daughterboard.core.cpu3.l1icache**
PV Cache.
Type: `PVCache`.

**FVP_VE_Cortex_R8x4.daughterboard.dram**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_R8x4.daughterboard.exclusive_monitor**
Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_VE_Cortex_R8x4.daughterboard.periph_clockdivider**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R8x4.daughterboard.pl310_l2cc**
ARM PrimeCell Level 2 Cache Controller (PL310).
Type: `PL310_L2CC`.

**FVP_VE_Cortex_R8x4.daughterboard.veinterruptmapper**
Interrupt Mapping peripheral (non-cascaded).
Type: `VEInterruptMapper`.

**FVP_VE_Cortex_R8x4.vemotherboard**
Model inspired by the ARM Versatile Express Motherboard for R profile.
Type: `VEMotherBoardR`.

**FVP_VE_Cortex_R8x4.vemotherboard.audioout**
SDL based Audio Output for PL041_AACI.
Type: `AudioOut_SDL`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_R8x4.vemotherboard.clock100Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R8x4.vemotherboard.clock24MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R8x4.vemotherboard.clock35MHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R8x4.vemotherboard.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R8x4.vemotherboard.clockCLCD**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R8x4.vemotherboard.dummy_ram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_R8x4.vemotherboard.dummy_usb**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_R8x4.vemotherboard.flash0**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

**FVP_VE_Cortex_R8x4.vemotherboard.flash1**

Intel Strata Flash J3 LISA+ model.
Type: `IntelStrataFlashJ3`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_R8x4.vemotherboard.flashloader0**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_VE_Cortex_R8x4.vemotherboard.flashloader1**

A device that can preload a gzipped image into flash at startup.
Type: `FlashLoader`.

**FVP_VE_Cortex_R8x4.vemotherboard.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_VE_Cortex_R8x4.vemotherboard.mmc**

Generic Multimedia Card.
Type: `MMC`.

**FVP_VE_Cortex_R8x4.vemotherboard.mmc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_R8x4.vemotherboard.mmc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_R8x4.vemotherboard.mmc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_R8x4.vemotherboard.mmc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_R8x4.vemotherboard.pl011_uart0**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_R8x4.vemotherboard.pl011_uart0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R8x4.vemotherboard.pl011_uart0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_R8x4.vemotherboard.pl011_uart0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_R8x4.vemotherboard.pl011_uart0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_R8x4.vemotherboard.pl011_uart0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_R8x4.vemotherboard.pl011_uart1**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_R8x4.vemotherboard.pl011_uart1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R8x4.vemotherboard.pl011_uart1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ClockTimerThread`.

**FVP_VE_Cortex_R8x4.vemotherboard.pl011_uart1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_R8x4.vemotherboard.pl011_uart1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_R8x4.vemotherboard.pl011_uart1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_R8x4.vemotherboard.pl011_uart2**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_R8x4.vemotherboard.pl011_uart2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R8x4.vemotherboard.pl011_uart2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_R8x4.vemotherboard.pl011_uart2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_R8x4.vemotherboard.pl011_uart2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_R8x4.vemotherboard.pl011_uart2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_R8x4.vemotherboard.pl011_uart3**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

**FVP_VE_Cortex_R8x4.vemotherboard.pl011_uart3.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R8x4.vemotherboard.pl011_uart3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_R8x4.vemotherboard.pl011_uart3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_R8x4.vemotherboard.pl011_uart3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_R8x4.vemotherboard.pl011_uart3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_R8x4.vemotherboard.pl011_uart4**

ARM PrimeCell UART(PL011).
Type: `PL011_Uart`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_R8x4.vemotherboard.pl011_uart4.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R8x4.vemotherboard.pl011_uart4.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_R8x4.vemotherboard.pl011_uart4.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_R8x4.vemotherboard.pl011_uart4.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_R8x4.vemotherboard.pl011_uart4.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_R8x4.vemotherboard.pl031_rtc**

ARM PrimeCell Real Time Clock(PL031).
Type: `PL031_RTC`.

**FVP_VE_Cortex_R8x4.vemotherboard.pl031_rtc.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_R8x4.vemotherboard.pl031_rtc.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_R8x4.vemotherboard.pl031_rtc.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_R8x4.vemotherboard.pl031_rtc.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_R8x4.vemotherboard.pl041_aaci**

ARM PrimeCell Advanced Audio CODEC Interface(PL041).
Type: `PL041_AACI`.

**FVP_VE_Cortex_R8x4.vemotherboard.pl041_aaci.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_R8x4.vemotherboard.pl041_aaci.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_R8x4.vemotherboard.pl041_aaci.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_R8x4.vemotherboard.pl041_aaci.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_R8x4.vemotherboard.pl050_kmi0**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_VE_Cortex_R8x4.vemotherboard.pl050_kmi0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R8x4.vemotherboard.pl050_kmi1**

ARM PrimeCell PS2 Keyboard/Mouse Interface(PL050).
Type: `PL050_KMI`.

**FVP_VE_Cortex_R8x4.vemotherboard.pl050_kmi1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R8x4.vemotherboard.pl111_clcd**

ARM PrimeCell Color LCD Controller(PL111).
Type: `PL111_CLCD`.

**FVP_VE_Cortex_R8x4.vemotherboard.pl111_clcd.pl11x_clcd**

Internal component used by PL110 and PL111 CLCD controllers.
Type: `PL11x_CLCD`.

**FVP_VE_Cortex_R8x4.vemotherboard.pl111_clcd.pl11x_clcd.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_R8x4.vemotherboard.pl111_clcd.pl11x_clcd.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_R8x4.vemotherboard.pl111_clcd.pl11x_clcd.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `SchedulerThread`.

**FVP_VE_Cortex_R8x4.vemotherboard.pl111_clcd.pl11x_clcd.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_R8x4.vemotherboard.pl180_mci**

ARM PrimeCell Multimedia Card Interface (PL180).
Type: `PL180_MCI`.

**FVP_VE_Cortex_R8x4.vemotherboard.ps2keyboard**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Keyboard`.

**FVP_VE_Cortex_R8x4.vemotherboard.ps2keyboard.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_R8x4.vemotherboard.ps2keyboard.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_R8x4.vemotherboard.ps2keyboard.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_R8x4.vemotherboard.ps2keyboard.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_R8x4.vemotherboard.ps2mouse**

Interface component, which takes the keypress/release signals from the Visualisation component and translates them into clocked PS2Data signals which can be routed to a PL050_KMI component.
Type: `PS2Mouse`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_R8x4.vemotherboard.ps2mouse.ps2_clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_VE_Cortex_R8x4.vemotherboard.ps2mouse.ps2_clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_R8x4.vemotherboard.ps2mouse.ps2_clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_R8x4.vemotherboard.ps2mouse.ps2_clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_R8x4.vemotherboard.psram**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_VE_Cortex_R8x4.vemotherboard.smsc_91c111**

SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_VE_Cortex_R8x4.vemotherboard.sp805_wdog**

ARM Watchdog Module(SP805).
Type: `SP805_Watchdog`.

**FVP_VE_Cortex_R8x4.vemotherboard.sp805_wdog.clocktimer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_R8x4.vemotherboard.sp805_wdog.clocktimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_R8x4.vemotherboard.sp805_wdog.clocktimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_R8x4.vemotherboard.sp805_wdog.clocktimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_R8x4.vemotherboard.sp810_sysctrl**

Only EB relevant functionalities are fully implemented.
Type: `SP810_SysCtrl`.

**FVP_VE_Cortex_R8x4.vemotherboard.sp810_sysctrl.clkdiv_clk0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R8x4.vemotherboard.sp810_sysctrl.clkdiv_clk1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R8x4.vemotherboard.sp810_sysctrl.clkdiv_clk2**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R8x4.vemotherboard.sp810_sysctrl.clkdiv_clk3**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_R8x4.vemotherboard.terminal_0**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_R8x4.vemotherboard.terminal_1**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_R8x4.vemotherboard.terminal_2**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_R8x4.vemotherboard.terminal_3**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_R8x4.vemotherboard.terminal_4**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_VE_Cortex_R8x4.vemotherboard.timer_0_1**
ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_VE_Cortex_R8x4.vemotherboard.timer_0_1.clk_div0**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R8x4.vemotherboard.timer_0_1.clk_div1**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_VE_Cortex_R8x4.vemotherboard.timer_0_1.counter0**
Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_VE_Cortex_R8x4.vemotherboard.timer_0_1.counter1**
Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_VE_Cortex_R8x4.vemotherboard.timer_2_3**
ARM Dual-Timer Module(SP804).
Type: `SP804_Timer`.

**FVP_VE_Cortex_R8x4.vemotherboard.timer_2_3.clk_div0**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**`FVP_VE_Cortex_R8x4.vemotherboard.timer_2_3.clk_div1`**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**`FVP_VE_Cortex_R8x4.vemotherboard.timer_2_3.counter0`**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**`FVP_VE_Cortex_R8x4.vemotherboard.timer_2_3.counter1`**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**`FVP_VE_Cortex_R8x4.vemotherboard.ve_sysregs`**

Type: `VE_SysRegs`.

**`FVP_VE_Cortex_R8x4.vemotherboard.video_ram`**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**`FVP_VE_Cortex_R8x4.vemotherboard.vis`**

Display window for VE using Visualisation library.
Type: `VEVisualisation`.

**`FVP_VE_Cortex_R8x4.vemotherboard.vis.recorder`**

Event recorder component for visualisation component (allows to playback and record interactive GUI sessions).
Type: `VisEventRecorder`.

**`FVP_VE_Cortex_R8x4.vemotherboard.vis.recorder.playbackDivider`**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**`FVP_VE_Cortex_R8x4.vemotherboard.vis.recorder.playbackTimer`**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_VE_Cortex_R8x4.vemotherboard.vis.recorder.playbackTimer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_VE_Cortex_R8x4.vemotherboard.vis.recorder.playbackTimer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_VE_Cortex_R8x4.vemotherboard.vis.recorder.playbackTimer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_VE_Cortex_R8x4.vemotherboard.vis.recorder.recordingDivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

# 16. MPS2 Platform FVPs

This chapter describes the MPS2 Platform FVPs contained in the FVP Standard Library.

For the MPS2 memory maps, see MPS2 - memory maps in the *Fast Models Reference Guide*.

## 16.1 FVP_MPS2_AEMv8M

List of instances in FVP_MPS2_AEMv8M.

### FVP_MPS2_AEMv8M instances

**FVP_MPS2_AEMv8M**

Type: `FVP_MPS2_AEMv8M`.

**FVP_MPS2_AEMv8M.clk25Mhz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_AEMv8M.clk25khz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_AEMv8M.cpu0**

ARM AEMv8M CT model.
Type: `ARM_AEMv8M`.

**FVP_MPS2_AEMv8M.cpu1**

ARM AEMv8M CT model.
Type: `ARM_AEMv8M`.

**FVP_MPS2_AEMv8M.fvp_mps2**

MPS2 DUT.
Type: `FVP_MPS2`.

**FVP_MPS2_AEMv8M.fvp_mps2.GPIO0**

ARM PrimeCell General Purpose Input/Output.
Type: `CMSDK_GPIO`.

**FVP_MPS2_AEMv8M.fvp_mps2.GPIO1**

ARM PrimeCell General Purpose Input/Output.
Type: `CMSDK_GPIO`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_AEMv8M.fvp_mps2.GPIO2**

ARM PrimeCell General Purpose Input/Output.
Type: CMSDK_GPIO.

**FVP_MPS2_AEMv8M.fvp_mps2.GPIO3**

ARM PrimeCell General Purpose Input/Output.
Type: CMSDK_GPIO.

**FVP_MPS2_AEMv8M.fvp_mps2.GPIO_connection_test**

Type: GPIO_Connection_Test.

**FVP_MPS2_AEMv8M.fvp_mps2.GPIO_connection_test.GPIO0_port_trans**

Type: GPIO_Port_Transfer.

**FVP_MPS2_AEMv8M.fvp_mps2.GPIO_connection_test.GPIO1_port_test**

Type: GPIO1_Connection_Test.

**FVP_MPS2_AEMv8M.fvp_mps2.PSRAM**

RAM device, can be dynamic or static ram.
Type: RAMDevice.

**FVP_MPS2_AEMv8M.fvp_mps2.PSRAM_M7**

RAM device, can be dynamic or static ram.
Type: RAMDevice.

**FVP_MPS2_AEMv8M.fvp_mps2.UART0**

ARM CMSDK UART Module.
Type: CMSDK_UART.

**FVP_MPS2_AEMv8M.fvp_mps2.UART0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: ClockDivider.

**FVP_MPS2_AEMv8M.fvp_mps2.UART1**

ARM CMSDK UART Module.
Type: CMSDK_UART.

**FVP_MPS2_AEMv8M.fvp_mps2.UART1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: ClockDivider.

**FVP_MPS2_AEMv8M.fvp_mps2.UART2**

ARM CMSDK UART Module.
Type: CMSDK_UART.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_AEMv8M.fvp_mps2.UART2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_AEMv8M.fvp_mps2.VGA_interface**

VGA display interface between main bus and visualisation.
Type: `MPS2_VGA`.

**FVP_MPS2_AEMv8M.fvp_mps2.ahb_ppc_iotss_expansion0**

IoT Subsystem Peripheral Protection Controller.
Type: `IoTSS_PeripheralProtectionController`.

**FVP_MPS2_AEMv8M.fvp_mps2.ahb_ppc_iotss_expansion1**

IoT Subsystem Peripheral Protection Controller.
Type: `IoTSS_PeripheralProtectionController`.

**FVP_MPS2_AEMv8M.fvp_mps2.apb_ppc_iotss_expansion0**

IoT Subsystem Peripheral Protection Controller.
Type: `IoTSS_PeripheralProtectionController`.

**FVP_MPS2_AEMv8M.fvp_mps2.apb_ppc_iotss_expansion1**

IoT Subsystem Peripheral Protection Controller.
Type: `IoTSS_PeripheralProtectionController`.

**FVP_MPS2_AEMv8M.fvp_mps2.apb_ppc_iotss_expansion2**

IoT Subsystem Peripheral Protection Controller.
Type: `IoTSS_PeripheralProtectionController`.

**FVP_MPS2_AEMv8M.fvp_mps2.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_AEMv8M.fvp_mps2.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_AEMv8M.fvp_mps2.cmsdk_sysctrl**

Cortex-M Simple System Control.
Type: `CMSDK_SysCtrl`.

**FVP_MPS2_AEMv8M.fvp_mps2.cmsdk_watchdog**

ARM Watchdog Module.
Type: `CMSDK_Watchdog`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_AEMv8M.fvp_mps2.cpu_wait_or_gate_0**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_AEMv8M.fvp_mps2.cpu_wait_or_gate_1**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_AEMv8M.fvp_mps2.dbgen_or_gate**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_AEMv8M.fvp_mps2.dma0**

ARM PrimeCell DMA Controller(PL080/081).
Type: `PL080_DMAC`.

**FVP_MPS2_AEMv8M.fvp_mps2.dma0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_MPS2_AEMv8M.fvp_mps2.dma0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_MPS2_AEMv8M.fvp_mps2.dma0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_MPS2_AEMv8M.fvp_mps2.dma0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_AEMv8M.fvp_mps2.dma0_idau_labeller**

Type: `LabellerIdauSecurity`.

**FVP_MPS2_AEMv8M.fvp_mps2.dma0_securitymodifier**

Type: `SecurityModifier`.

**FVP_MPS2_AEMv8M.fvp_mps2.dma1**

ARM PrimeCell DMA Controller(PL080/081).

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `PL080_DMAC`.

**FVP_MPS2_AEMv8M.fvp_mps2.dma1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_MPS2_AEMv8M.fvp_mps2.dma1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_MPS2_AEMv8M.fvp_mps2.dma1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_MPS2_AEMv8M.fvp_mps2.dma1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_AEMv8M.fvp_mps2.dma1_idau_labeller**

Type: `LabellerIdauSecurity`.

**FVP_MPS2_AEMv8M.fvp_mps2.dma1_securitymodifier**

Type: `SecurityModifier`.

**FVP_MPS2_AEMv8M.fvp_mps2.dma2**

ARM PrimeCell DMA Controller(PL080/081).
Type: `PL080_DMAC`.

**FVP_MPS2_AEMv8M.fvp_mps2.dma2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_AEMv8M.fvp_mps2.dma2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_MPS2_AEMv8M.fvp_mps2.dma2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_MPS2_AEMv8M.fvp_mps2.dma2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_AEMv8M.fvp_mps2.dma2_idau_labeller**

Type: `LabellerIdauSecurity`.

**FVP_MPS2_AEMv8M.fvp_mps2.dma2_securitymodifier**

Type: `SecurityModifier`.

**FVP_MPS2_AEMv8M.fvp_mps2.dma3**

ARM PrimeCell DMA Controller(PL080/081).
Type: `PL080_DMAC`.

**FVP_MPS2_AEMv8M.fvp_mps2.dma3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_MPS2_AEMv8M.fvp_mps2.dma3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_MPS2_AEMv8M.fvp_mps2.dma3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_AEMv8M.fvp_mps2.dma3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_AEMv8M.fvp_mps2.dma3_idau_labeller**

Type: `LabellerIdauSecurity`.

**FVP_MPS2_AEMv8M.fvp_mps2.dma3_securitymodifier**

Type: `SecurityModifier`.

**FVP_MPS2_AEMv8M.fvp_mps2.exclusive_monitor_psram**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_AEMv8M.fvp_mps2.exclusive_monitor_psram_iotss**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_AEMv8M.fvp_mps2.exclusive_monitor_switch_extra_psram_iotss**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_AEMv8M.fvp_mps2.exclusive_monitor_switch_extra_psram_mps2**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_AEMv8M.fvp_mps2.exclusive_monitor_zbtsram1**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_AEMv8M.fvp_mps2.exclusive_monitor_zbtsram2**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_AEMv8M.fvp_mps2.exclusive_squasher**

Squashes the exclusive attribute on bus transactions.
Type: `PVBusExclusiveSquasher`.

**FVP_MPS2_AEMv8M.fvp_mps2.fpga_sysctrl**

FPGA SysCtrl timers LEDs and switches.
Type: `FPGA_SysCtrl`.

**FVP_MPS2_AEMv8M.fvp_mps2.fpga_sysctrl.callBack100HzCounter**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_AEMv8M.fvp_mps2.fpga_sysctrl.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_AEMv8M.fvp_mps2.gpio_0_or_2**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_AEMv8M.fvp_mps2.gpio_1_or_3**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_AEMv8M.fvp_mps2.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_MPS2_AEMv8M.fvp_mps2.mem_switch_extra_psram_iotss**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_AEMv8M.fvp_mps2.mem_switch_extra_psram_mps2**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_AEMv8M.fvp_mps2.mpc_iotss_ssram1**

IoT Subsystem Memory Protection Controller.
Type: `IoTSS_MemoryProtectionController`.

**FVP_MPS2_AEMv8M.fvp_mps2.mpc_iotss_ssram1.gating_disabled_thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_AEMv8M.fvp_mps2.mpc_iotss_ssram2**

IoT Subsystem Memory Protection Controller.
Type: `IoTSS_MemoryProtectionController`.

**FVP_MPS2_AEMv8M.fvp_mps2.mpc_iotss_ssram2.gating_disabled_thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_AEMv8M.fvp_mps2.mpc_iotss_ssram3**

IoT Subsystem Memory Protection Controller.
Type: `IoTSS_MemoryProtectionController`.

**FVP_MPS2_AEMv8M.fvp_mps2.mpc_iotss_ssram3.gating_disabled_thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_AEMv8M.fvp_mps2.mps2_audio**

MPS2 Audio.
Type: `MPS2_Audio`.

**FVP_MPS2_AEMv8M.fvp_mps2.mps2_cmsdk_dualtimer**

ARM Dual-Timer Module.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `CMSDK_DualTimer`.

**FVP_MPS2_AEMv8M.fvp_mps2.mps2_cmsdk_dualtimer.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_AEMv8M.fvp_mps2.mps2_cmsdk_dualtimer.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_AEMv8M.fvp_mps2.mps2_cmsdk_dualtimer.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_AEMv8M.fvp_mps2.mps2_cmsdk_dualtimer.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_AEMv8M.fvp_mps2.mps2_exclusive_monitor_zbtsram1**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_AEMv8M.fvp_mps2.mps2_exclusive_monitor_zbtsram2**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_AEMv8M.fvp_mps2.mps2_lcd**

MPS2 LCD I2C interface.
Type: `MPS2_LCD`.

**FVP_MPS2_AEMv8M.fvp_mps2.mps2_mpc_iotss_ssram1**

IoT Subsystem Memory Protection Controller.
Type: `IoTSS_MemoryProtectionController`.

**FVP_MPS2_AEMv8M.fvp_mps2.mps2_mpc_iotss_ssram1.gating_disabled_thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_AEMv8M.fvp_mps2.mps2_secure_control_register_block**

MPS2 Secure Control Register Block.
Type: `MPS2_SecureCtrl`.

**FVP_MPS2_AEMv8M.fvp_mps2.mps2_timer0**

ARM Timer Module.
Type: `CMSDK_Timer`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_AEMv8M.fvp_mps2.mps2_timer0.clk_div**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_AEMv8M.fvp_mps2.mps2_timer0.counter**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_AEMv8M.fvp_mps2.mps2_timer1**

ARM Timer Module.
Type: `CMSDK_Timer`.

**FVP_MPS2_AEMv8M.fvp_mps2.mps2_timer1.clk_div**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_AEMv8M.fvp_mps2.mps2_timer1.counter**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_AEMv8M.fvp_mps2.mps2_visualisation**

Display window for MPS2 using Visualisation library.
Type: `MPS2_Visualisation`.

**FVP_MPS2_AEMv8M.fvp_mps2.niden_or_gate**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_AEMv8M.fvp_mps2.nmi_or_gate**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_AEMv8M.fvp_mps2.pl022_ssp_mps2**

ARM PrimeCell Synchronous Serial Port(PL022).
Type: `PL022_SSP_MPS2`.

**FVP_MPS2_AEMv8M.fvp_mps2.pl022_ssp_mps2.prescaler**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_AEMv8M.fvp_mps2.platform_bus_switch**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_AEMv8M.fvp_mps2.platform_switch_dma0**
Allow transactions to be routed arbitrarily.
Type: PVBusRouter.

**FVP_MPS2_AEMv8M.fvp_mps2.platform_switch_dma1**
Allow transactions to be routed arbitrarily.
Type: PVBusRouter.

**FVP_MPS2_AEMv8M.fvp_mps2.signal_router**
Signal router.
Type: SignalRouter.

**FVP_MPS2_AEMv8M.fvp_mps2.smsc_91c111**
SMSC 91C111 ethernet controller.
Type: SMSC_91C111.

**FVP_MPS2_AEMv8M.fvp_mps2.spiden_or_gate**
Or Gate.
Type: OrGate.

**FVP_MPS2_AEMv8M.fvp_mps2.spniden_or_gate**
Or Gate.
Type: OrGate.

**FVP_MPS2_AEMv8M.fvp_mps2.sse200**
SSE-200 subsystem.
Type: SSE200.

**FVP_MPS2_AEMv8M.fvp_mps2.sse200.acg_cpu0**
IoT Subsystem Access Control Gate.
Type: IoTSS_AccessControlGate.

**FVP_MPS2_AEMv8M.fvp_mps2.sse200.acg_cpu1**
IoT Subsystem Access Control Gate.
Type: IoTSS_AccessControlGate.

**FVP_MPS2_AEMv8M.fvp_mps2.sse200.acg_sram0**
IoT Subsystem Access Control Gate.
Type: IoTSS_AccessControlGate.

**FVP_MPS2_AEMv8M.fvp_mps2.sse200.acg_sram1**
IoT Subsystem Access Control Gate.
Type: IoTSS_AccessControlGate.

**FVP_MPS2_AEMv8M.fvp_mps2.sse200.acg_sram2**
IoT Subsystem Access Control Gate.
Type: IoTSS_AccessControlGate.

**FVP_MPS2_AEMv8M.fvp_mps2.sse200.acg_sram3**
IoT Subsystem Access Control Gate.
Type: IoTSS_AccessControlGate.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_AEMv8M.fvp_mps2.sse200.apb_ppc_iotss_subsystem0**

IoT Subsystem Peripheral Protection Controller.
Type: `IoTSS_PeripheralProtectionController`.

**FVP_MPS2_AEMv8M.fvp_mps2.sse200.apb_ppc_iotss_subsystem1**

IoT Subsystem Peripheral Protection Controller.
Type: `IoTSS_PeripheralProtectionController`.

**FVP_MPS2_AEMv8M.fvp_mps2.sse200.clock32kHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_AEMv8M.fvp_mps2.sse200.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_AEMv8M.fvp_mps2.sse200.cmsdk_dualtimer**

ARM Dual-Timer Module.
Type: `CMSDK_DualTimer`.

**FVP_MPS2_AEMv8M.fvp_mps2.sse200.cmsdk_dualtimer.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_AEMv8M.fvp_mps2.sse200.cmsdk_dualtimer.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_AEMv8M.fvp_mps2.sse200.cmsdk_dualtimer.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_AEMv8M.fvp_mps2.sse200.cmsdk_dualtimer.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_AEMv8M.fvp_mps2.sse200.cordio_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_AEMv8M.fvp_mps2.sse200.cpu0core_ppu**
ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_AEMv8M.fvp_mps2.sse200.cpu0dbg_ppu**
ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_AEMv8M.fvp_mps2.sse200.cpu1core_ppu**
ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_AEMv8M.fvp_mps2.sse200.cpu1dbg_ppu**
ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_AEMv8M.fvp_mps2.sse200.crypto_ppu**
ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_AEMv8M.fvp_mps2.sse200.dbg_ppu**
ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_AEMv8M.fvp_mps2.sse200.exclusive_monitor_iotss_internal_sram0**
Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_AEMv8M.fvp_mps2.sse200.exclusive_monitor_iotss_internal_sram1**
Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_AEMv8M.fvp_mps2.sse200.exclusive_monitor_iotss_internal_sram2**
Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_AEMv8M.fvp_mps2.sse200.exclusive_monitor_iotss_internal_sram3**
Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_AEMv8M.fvp_mps2.sse200.exclusive_squasher**
Squashes the exclusive attribute on bus transactions.
Type: `PVBusExclusiveSquasher`.

**FVP_MPS2_AEMv8M.fvp_mps2.sse200.idau_labeller**
Type: `LabellerIdauSecurity`.

**FVP_MPS2_AEMv8M.fvp_mps2.sse200.internal_msniper_tcm**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_AEMv8M.fvp_mps2.sse200.iotss_cpuidentity**
IoT Subsystem CPU_IDENTITY registers.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `IoTSS_CPUIdentity`.

**FVP_MPS2_AEMv8M.fvp_mps2.sse200.iotss_internal_sram0**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_AEMv8M.fvp_mps2.sse200.iotss_internal_sram1**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_AEMv8M.fvp_mps2.sse200.iotss_internal_sram2**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_AEMv8M.fvp_mps2.sse200.iotss_internal_sram3**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_AEMv8M.fvp_mps2.sse200.iotss_systemcontrol**

IoT Subsystem System Control registers.
Type: `IoTSS_SystemControl`.

**FVP_MPS2_AEMv8M.fvp_mps2.sse200.iotss_systeminfo**

IoT Subsystem System Information registers.
Type: `IoTSS_SystemInfo`.

**FVP_MPS2_AEMv8M.fvp_mps2.sse200.mem_switch_internal_msniper_tcm**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_AEMv8M.fvp_mps2.sse200.mem_switch_internal_sram**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_AEMv8M.fvp_mps2.sse200.mem_switch_internal_sram_mpc**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_AEMv8M.fvp_mps2.sse200.mem_switch_ppu**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_AEMv8M.fvp_mps2.sse200.mhu0**

IoT Subsystem Message Handling Unit.
Type: `IoTSS_MessageHandlingUnit`.

**FVP_MPS2_AEMv8M.fvp_mps2.sse200.mhu1**

IoT Subsystem Message Handling Unit.
Type: `IoTSS_MessageHandlingUnit`.

**FVP_MPS2_AEMv8M.fvp_mps2.sse200.mpc_iotss_internal_sram0**

IoT Subsystem Memory Protection Controller.
Type: `IoTSS_MemoryProtectionController`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_AEMv8M.fvp_mps2.sse200.mpc_iotss_internal_sram0.gating_disabled_thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_AEMv8M.fvp_mps2.sse200.mpc_iotss_internal_sram1**

IoT Subsystem Memory Protection Controller.
Type: `IoTSS_MemoryProtectionController`.

**FVP_MPS2_AEMv8M.fvp_mps2.sse200.mpc_iotss_internal_sram1.gating_disabled_thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_AEMv8M.fvp_mps2.sse200.mpc_iotss_internal_sram2**

IoT Subsystem Memory Protection Controller.
Type: `IoTSS_MemoryProtectionController`.

**FVP_MPS2_AEMv8M.fvp_mps2.sse200.mpc_iotss_internal_sram2.gating_disabled_thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_AEMv8M.fvp_mps2.sse200.mpc_iotss_internal_sram3**

IoT Subsystem Memory Protection Controller.
Type: `IoTSS_MemoryProtectionController`.

**FVP_MPS2_AEMv8M.fvp_mps2.sse200.mpc_iotss_internal_sram3.gating_disabled_thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_AEMv8M.fvp_mps2.sse200.nmi_or_gate**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_AEMv8M.fvp_mps2.sse200.nonsecure_watchdog**

ARM Watchdog Module.
Type: `CMSDK_Watchdog`.

**FVP_MPS2_AEMv8M.fvp_mps2.sse200.ram0_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_AEMv8M.fvp_mps2.sse200.ram1_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_AEMv8M.fvp_mps2.sse200.ram2_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_AEMv8M.fvp_mps2.sse200.ram3_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_AEMv8M.fvp_mps2.sse200.s32k_timer**

ARM Timer Module.
Type: `CMSDK_Timer`.

**FVP_MPS2_AEMv8M.fvp_mps2.sse200.s32k_timer.clk_div**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_AEMv8M.fvp_mps2.sse200.s32k_timer.counter**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_AEMv8M.fvp_mps2.sse200.s32k_watchdog**

ARM Watchdog Module.
Type: `CMSDK_Watchdog`.

**FVP_MPS2_AEMv8M.fvp_mps2.sse200.secure_control_register_block**

MPS2 Secure Control Register Block.
Type: `MPS2_SecureCtrl`.

**FVP_MPS2_AEMv8M.fvp_mps2.sse200.secure_watchdog**

ARM Watchdog Module.
Type: `CMSDK_Watchdog`.

**FVP_MPS2_AEMv8M.fvp_mps2.sse200.signal_router**

Signal router.
Type: `SignalRouter`.

**FVP_MPS2_AEMv8M.fvp_mps2.sse200.sys_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_AEMv8M.fvp_mps2.sse200.timer0**

ARM Timer Module.
Type: `CMSDK_Timer`.

**FVP_MPS2_AEMv8M.fvp_mps2.sse200.timer0.clk_div**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_AEMv8M.fvp_mps2.sse200.timer0.counter**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_AEMv8M.fvp_mps2.sse200.timer1**

ARM Timer Module.
Type: `CMSDK_Timer`.

**FVP_MPS2_AEMv8M.fvp_mps2.sse200.timer1.clk_div**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_AEMv8M.fvp_mps2.sse200.timer1.counter**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_AEMv8M.fvp_mps2.ssram1**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_AEMv8M.fvp_mps2.ssram2**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_AEMv8M.fvp_mps2.stub0**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_AEMv8M.fvp_mps2.stub1**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_AEMv8M.fvp_mps2.stub_i2c1**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_AEMv8M.fvp_mps2.stub_i2s**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_AEMv8M.fvp_mps2.stub_spi0**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_AEMv8M.fvp_mps2.stub_spi2**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_AEMv8M.fvp_mps2.svos_dualtimer**

Type: `SVOS_DualTimer`.

**FVP_MPS2_AEMv8M.fvp_mps2.svos_dualtimer.svos_dualtimer0**

ARM Dual-Timer Module.
Type: `CMSDK_DualTimer`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_AEMv8M.fvp_mps2.svos_dualtimer.svos_dualtimer0.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_AEMv8M.fvp_mps2.svos_dualtimer.svos_dualtimer0.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_AEMv8M.fvp_mps2.svos_dualtimer.svos_dualtimer0.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_AEMv8M.fvp_mps2.svos_dualtimer.svos_dualtimer0.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_AEMv8M.fvp_mps2.svos_dualtimer.svos_dualtimer1**

ARM Dual-Timer Module.
Type: `CMSDK_DualTimer`.

**FVP_MPS2_AEMv8M.fvp_mps2.svos_dualtimer.svos_dualtimer1.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_AEMv8M.fvp_mps2.svos_dualtimer.svos_dualtimer1.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_AEMv8M.fvp_mps2.svos_dualtimer.svos_dualtimer1.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_AEMv8M.fvp_mps2.svos_dualtimer.svos_dualtimer1.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_AEMv8M.fvp_mps2.svos_dualtimer.svos_dualtimer2**

ARM Dual-Timer Module.
Type: `CMSDK_DualTimer`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_AEMv8M.fvp_mps2.svos_dualtimer.svos_dualtimer2.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_AEMv8M.fvp_mps2.svos_dualtimer.svos_dualtimer2.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_AEMv8M.fvp_mps2.svos_dualtimer.svos_dualtimer2.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_AEMv8M.fvp_mps2.svos_dualtimer.svos_dualtimer2.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_AEMv8M.fvp_mps2.svos_dualtimer.svos_dualtimer3**

ARM Dual-Timer Module.
Type: `CMSDK_DualTimer`.

**FVP_MPS2_AEMv8M.fvp_mps2.svos_dualtimer.svos_dualtimer3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_AEMv8M.fvp_mps2.svos_dualtimer.svos_dualtimer3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_AEMv8M.fvp_mps2.svos_dualtimer.svos_dualtimer3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_AEMv8M.fvp_mps2.svos_dualtimer.svos_dualtimer3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_AEMv8M.fvp_mps2.switch_PSRAM_M7**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_AEMv8M.fvp_mps2.switch_svos_dualtimer**
Allow transactions to be routed arbitrarily.
Type: PVBusRouter.

**FVP_MPS2_AEMv8M.fvp_mps2.telnetterminal0**
Telnet terminal interface.
Type: TelnetTerminal.

**FVP_MPS2_AEMv8M.fvp_mps2.telnetterminal1**
Telnet terminal interface.
Type: TelnetTerminal.

**FVP_MPS2_AEMv8M.fvp_mps2.telnetterminal2**
Telnet terminal interface.
Type: TelnetTerminal.

**FVP_MPS2_AEMv8M.fvp_mps2.touchscreen_interface**
MPS2 Touch Screen.
Type: MPS2_TouchScreen.

**FVP_MPS2_AEMv8M.fvp_mps2.uart_overflows_or_gate**
Or Gate.
Type: OrGate.

## 16.2 FVP_MPS2_Cortex-M0

List of instances in FVP_MPS2_Cortex-M0.

### FVP_MPS2_Cortex-M0 instances

**FVP_MPS2_Cortex_M0**
Type: FVP_MPS2_Cortex_M0.

**FVP_MPS2_Cortex_M0.armcortexm0ct**
ARM CORTEXM0 CT model.
Type: ARM_Cortex-M0.

**FVP_MPS2_Cortex_M0.clk25Mhz**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: ClockDivider.

**FVP_MPS2_Cortex_M0.clk25khz**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: ClockDivider.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M0.fvp_mps2**

MPS2 DUT.
Type: `FVP_MPS2`.

**FVP_MPS2_Cortex_M0.fvp_mps2.GPIO0**

ARM PrimeCell General Purpose Input/Output.
Type: `CMSDK_GPIO`.

**FVP_MPS2_Cortex_M0.fvp_mps2.GPIO1**

ARM PrimeCell General Purpose Input/Output.
Type: `CMSDK_GPIO`.

**FVP_MPS2_Cortex_M0.fvp_mps2.GPIO2**

ARM PrimeCell General Purpose Input/Output.
Type: `CMSDK_GPIO`.

**FVP_MPS2_Cortex_M0.fvp_mps2.GPIO3**

ARM PrimeCell General Purpose Input/Output.
Type: `CMSDK_GPIO`.

**FVP_MPS2_Cortex_M0.fvp_mps2.GPIO_connection_test**

Type: `GPIO_Connection_Test`.

**FVP_MPS2_Cortex_M0.fvp_mps2.GPIO_connection_test.GPIO0_port_trans**

Type: `GPIO_Port_Transfer`.

**FVP_MPS2_Cortex_M0.fvp_mps2.GPIO_connection_test.GPIO1_port_test**

Type: `GPIO1_Connection_Test`.

**FVP_MPS2_Cortex_M0.fvp_mps2.PSRAM**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M0.fvp_mps2.PSRAM_M7**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M0.fvp_mps2.UART0**

ARM CMSDK UART Module.
Type: `CMSDK_UART`.

**FVP_MPS2_Cortex_M0.fvp_mps2.UART0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M0.fvp_mps2.UART1**

ARM CMSDK UART Module.
Type: `CMSDK_UART`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M0.fvp_mps2.UART1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M0.fvp_mps2.UART2**

ARM CMSDK UART Module.
Type: `CMSDK_UART`.

**FVP_MPS2_Cortex_M0.fvp_mps2.UART2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M0.fvp_mps2.VGA_interface**

VGA display interface between main bus and visualisation.
Type: `MPS2_VGA`.

**FVP_MPS2_Cortex_M0.fvp_mps2.ahb_ppc_iotss_expansion0**

IoT Subsystem Peripheral Protection Controller.
Type: `IoTSS_PeripheralProtectionController`.

**FVP_MPS2_Cortex_M0.fvp_mps2.ahb_ppc_iotss_expansion1**

IoT Subsystem Peripheral Protection Controller.
Type: `IoTSS_PeripheralProtectionController`.

**FVP_MPS2_Cortex_M0.fvp_mps2.apb_ppc_iotss_expansion0**

IoT Subsystem Peripheral Protection Controller.
Type: `IoTSS_PeripheralProtectionController`.

**FVP_MPS2_Cortex_M0.fvp_mps2.apb_ppc_iotss_expansion1**

IoT Subsystem Peripheral Protection Controller.
Type: `IoTSS_PeripheralProtectionController`.

**FVP_MPS2_Cortex_M0.fvp_mps2.apb_ppc_iotss_expansion2**

IoT Subsystem Peripheral Protection Controller.
Type: `IoTSS_PeripheralProtectionController`.

**FVP_MPS2_Cortex_M0.fvp_mps2.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M0.fvp_mps2.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M0.fvp_mps2.cmsdk_sysctrl**

Cortex-M Simple System Control.
Type: `CMSDK_SysCtrl`.

**FVP_MPS2_Cortex_M0.fvp_mps2.cmsdk_watchdog**

ARM Watchdog Module.
Type: `CMSDK_Watchdog`.

**FVP_MPS2_Cortex_M0.fvp_mps2.cpu_wait_or_gate_0**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_Cortex_M0.fvp_mps2.cpu_wait_or_gate_1**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_Cortex_M0.fvp_mps2.dbgen_or_gate**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_Cortex_M0.fvp_mps2.dma0**

ARM PrimeCell DMA Controller(PL080/081).
Type: `PL080_DMAC`.

**FVP_MPS2_Cortex_M0.fvp_mps2.dma0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_MPS2_Cortex_M0.fvp_mps2.dma0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_MPS2_Cortex_M0.fvp_mps2.dma0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M0.fvp_mps2.dma0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M0.fvp_mps2.dma0_idau_labeller**

Type: `LabellerIdauSecurity`.

**FVP_MPS2_Cortex_M0.fvp_mps2.dma0_securitymodifier**

Type: `SecurityModifier`.

**FVP_MPS2_Cortex_M0.fvp_mps2.dma1**

ARM PrimeCell DMA Controller(PL080/081).
Type: `PL080_DMAC`.

**FVP_MPS2_Cortex_M0.fvp_mps2.dma1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_MPS2_Cortex_M0.fvp_mps2.dma1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_MPS2_Cortex_M0.fvp_mps2.dma1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_MPS2_Cortex_M0.fvp_mps2.dma1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M0.fvp_mps2.dma1_idau_labeller**

Type: `LabellerIdauSecurity`.

**FVP_MPS2_Cortex_M0.fvp_mps2.dma1_securitymodifier**

Type: `SecurityModifier`.

**FVP_MPS2_Cortex_M0.fvp_mps2.dma2**

ARM PrimeCell DMA Controller(PL080/081).
Type: `PL080_DMAC`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M0.fvp_mps2.dma2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_MPS2_Cortex_M0.fvp_mps2.dma2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_MPS2_Cortex_M0.fvp_mps2.dma2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_MPS2_Cortex_M0.fvp_mps2.dma2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M0.fvp_mps2.dma2_idau_labeller**

Type: `LabellerIdauSecurity`.

**FVP_MPS2_Cortex_M0.fvp_mps2.dma2_securitymodifier**

Type: `SecurityModifier`.

**FVP_MPS2_Cortex_M0.fvp_mps2.dma3**

ARM PrimeCell DMA Controller(PL080/081).
Type: `PL080_DMAC`.

**FVP_MPS2_Cortex_M0.fvp_mps2.dma3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_MPS2_Cortex_M0.fvp_mps2.dma3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_MPS2_Cortex_M0.fvp_mps2.dma3.timer.timer.thread**
A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_MPS2_Cortex_M0.fvp_mps2.dma3.timer.timer.thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M0.fvp_mps2.dma3_idau_labeller**
Type: `LabellerIdauSecurity`.

**FVP_MPS2_Cortex_M0.fvp_mps2.dma3_securitymodifier**
Type: `SecurityModifier`.

**FVP_MPS2_Cortex_M0.fvp_mps2.exclusive_monitor_psram**
Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_Cortex_M0.fvp_mps2.exclusive_monitor_psram_iotss**
Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_Cortex_M0.fvp_mps2.exclusive_monitor_switch_extra_psram_iotss**
Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M0.fvp_mps2.exclusive_monitor_switch_extra_psram_mps2**
Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M0.fvp_mps2.exclusive_monitor_zbtsram1**
Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_Cortex_M0.fvp_mps2.exclusive_monitor_zbtsram2**
Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_Cortex_M0.fvp_mps2.exclusive_squasher**
Squashes the exclusive attribute on bus transactions.
Type: `PVBusExclusiveSquasher`.

**FVP_MPS2_Cortex_M0.fvp_mps2.fpga_sysctrl**
FPGA SysCtrl timers LEDs and switches.
Type: `FPGA_SysCtrl`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M0.fvp_mps2.fpga_sysctrl.callBack100HzCounter**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M0.fvp_mps2.fpga_sysctrl.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M0.fvp_mps2.gpio_0_or_2**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_Cortex_M0.fvp_mps2.gpio_1_or_3**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_Cortex_M0.fvp_mps2.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_MPS2_Cortex_M0.fvp_mps2.mem_switch_extra_psram_iotss**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M0.fvp_mps2.mem_switch_extra_psram_mps2**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M0.fvp_mps2.mpc_iotss_ssram1**

IoT Subsystem Memory Protection Controller.
Type: `IoTSS_MemoryProtectionController`.

**FVP_MPS2_Cortex_M0.fvp_mps2.mpc_iotss_ssram1.gating_disabled_thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M0.fvp_mps2.mpc_iotss_ssram2**

IoT Subsystem Memory Protection Controller.
Type: `IoTSS_MemoryProtectionController`.

**FVP_MPS2_Cortex_M0.fvp_mps2.mpc_iotss_ssram2.gating_disabled_thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M0.fvp_mps2.mpc_iotss_ssram3**

IoT Subsystem Memory Protection Controller.
Type: `IoTSS_MemoryProtectionController`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M0.fvp_mps2.mpc_iotss_ssram3.gating_disabled_thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M0.fvp_mps2.mps2_audio**

MPS2 Audio.
Type: `MPS2_Audio`.

**FVP_MPS2_Cortex_M0.fvp_mps2.mps2_cmsdk_dualtimer**

ARM Dual-Timer Module.
Type: `CMSDK_DualTimer`.

**FVP_MPS2_Cortex_M0.fvp_mps2.mps2_cmsdk_dualtimer.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M0.fvp_mps2.mps2_cmsdk_dualtimer.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M0.fvp_mps2.mps2_cmsdk_dualtimer.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M0.fvp_mps2.mps2_cmsdk_dualtimer.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M0.fvp_mps2.mps2_exclusive_monitor_zbtsram1**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_Cortex_M0.fvp_mps2.mps2_exclusive_monitor_zbtsram2**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_Cortex_M0.fvp_mps2.mps2_lcd**

MPS2 LCD I2C interface.
Type: `MPS2_LCD`.

**FVP_MPS2_Cortex_M0.fvp_mps2.mps2_mpc_iotss_ssram1**

IoT Subsystem Memory Protection Controller.
Type: `IoTSS_MemoryProtectionController`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M0.fvp_mps2.mps2_mpc_iotss_ssram1.gating_disabled_thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M0.fvp_mps2.mps2_secure_control_register_block**

MPS2 Secure Control Register Block.
Type: `MPS2_SecureCtrl`.

**FVP_MPS2_Cortex_M0.fvp_mps2.mps2_timer0**

ARM Timer Module.
Type: `CMSDK_Timer`.

**FVP_MPS2_Cortex_M0.fvp_mps2.mps2_timer0.clk_div**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M0.fvp_mps2.mps2_timer0.counter**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M0.fvp_mps2.mps2_timer1**

ARM Timer Module.
Type: `CMSDK_Timer`.

**FVP_MPS2_Cortex_M0.fvp_mps2.mps2_timer1.clk_div**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M0.fvp_mps2.mps2_timer1.counter**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M0.fvp_mps2.mps2_visualisation**

Display window for MPS2 using Visualisation library.
Type: `MPS2_Visualisation`.

**FVP_MPS2_Cortex_M0.fvp_mps2.niden_or_gate**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_Cortex_M0.fvp_mps2.nmi_or_gate**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_Cortex_M0.fvp_mps2.pl022_ssp_mps2**

ARM PrimeCell Synchronous Serial Port(PL022).

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `PL022_SSP_MPS2`.

**`FVP_MPS2_Cortex_M0.fvp_mps2.pl022_ssp_mps2.prescaler`**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**`FVP_MPS2_Cortex_M0.fvp_mps2.platform_bus_switch`**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**`FVP_MPS2_Cortex_M0.fvp_mps2.platform_switch_dma0`**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**`FVP_MPS2_Cortex_M0.fvp_mps2.platform_switch_dma1`**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**`FVP_MPS2_Cortex_M0.fvp_mps2.signal_router`**

Signal router.
Type: `SignalRouter`.

**`FVP_MPS2_Cortex_M0.fvp_mps2.smsc_91c111`**

SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**`FVP_MPS2_Cortex_M0.fvp_mps2.spiden_or_gate`**

Or Gate.
Type: `OrGate`.

**`FVP_MPS2_Cortex_M0.fvp_mps2.spniden_or_gate`**

Or Gate.
Type: `OrGate`.

**`FVP_MPS2_Cortex_M0.fvp_mps2.sse200`**

SSE-200 subsystem.
Type: `SSE200`.

**`FVP_MPS2_Cortex_M0.fvp_mps2.sse200.acg_cpu0`**

IoT Subsystem Access Control Gate.
Type: `IoTSS_AccessControlGate`.

**`FVP_MPS2_Cortex_M0.fvp_mps2.sse200.acg_cpu1`**

IoT Subsystem Access Control Gate.
Type: `IoTSS_AccessControlGate`.

**`FVP_MPS2_Cortex_M0.fvp_mps2.sse200.acg_sram0`**

IoT Subsystem Access Control Gate.
Type: `IoTSS_AccessControlGate`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M0.fvp_mps2.sse200.acg_sram1**

IoT Subsystem Access Control Gate.
Type: `IoTSS_AccessControlGate`.

**FVP_MPS2_Cortex_M0.fvp_mps2.sse200.acg_sram2**

IoT Subsystem Access Control Gate.
Type: `IoTSS_AccessControlGate`.

**FVP_MPS2_Cortex_M0.fvp_mps2.sse200.acg_sram3**

IoT Subsystem Access Control Gate.
Type: `IoTSS_AccessControlGate`.

**FVP_MPS2_Cortex_M0.fvp_mps2.sse200.apb_ppc_iotss_subsystem0**

IoT Subsystem Peripheral Protection Controller.
Type: `IoTSS_PeripheralProtectionController`.

**FVP_MPS2_Cortex_M0.fvp_mps2.sse200.apb_ppc_iotss_subsystem1**

IoT Subsystem Peripheral Protection Controller.
Type: `IoTSS_PeripheralProtectionController`.

**FVP_MPS2_Cortex_M0.fvp_mps2.sse200.clock32kHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M0.fvp_mps2.sse200.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M0.fvp_mps2.sse200.cmsdk_dualtimer**

ARM Dual-Timer Module.
Type: `CMSDK_DualTimer`.

**FVP_MPS2_Cortex_M0.fvp_mps2.sse200.cmsdk_dualtimer.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M0.fvp_mps2.sse200.cmsdk_dualtimer.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M0.fvp_mps2.sse200.cmsdk_dualtimer.counter0**
Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M0.fvp_mps2.sse200.cmsdk_dualtimer.counter1**
Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M0.fvp_mps2.sse200.cordio_ppu**
ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M0.fvp_mps2.sse200.cpu0core_ppu**
ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M0.fvp_mps2.sse200.cpu0dbg_ppu**
ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M0.fvp_mps2.sse200.cpu1core_ppu**
ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M0.fvp_mps2.sse200.cpu1dbg_ppu**
ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M0.fvp_mps2.sse200.crypto_ppu**
ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M0.fvp_mps2.sse200.dbg_ppu**
ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M0.fvp_mps2.sse200.exclusive_monitor_iotss_internal_sram0**
Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_Cortex_M0.fvp_mps2.sse200.exclusive_monitor_iotss_internal_sram1**
Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_Cortex_M0.fvp_mps2.sse200.exclusive_monitor_iotss_internal_sram2**
Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_Cortex_M0.fvp_mps2.sse200.exclusive_monitor_iotss_internal_sram3**
Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M0.fvp_mps2.sse200.exclusive_squasher**

Squashes the exclusive attribute on bus transactions.
Type: `PVBusExclusiveSquasher`.

**FVP_MPS2_Cortex_M0.fvp_mps2.sse200.idau_labeller**

Type: `LabellerIdauSecurity`.

**FVP_MPS2_Cortex_M0.fvp_mps2.sse200.internal_msniper_tcm**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M0.fvp_mps2.sse200.iotss_cpuidentity**

IoT Subsystem CPU_IDENTITY registers.
Type: `IoTSS_CPUIdentity`.

**FVP_MPS2_Cortex_M0.fvp_mps2.sse200.iotss_internal_sram0**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M0.fvp_mps2.sse200.iotss_internal_sram1**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M0.fvp_mps2.sse200.iotss_internal_sram2**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M0.fvp_mps2.sse200.iotss_internal_sram3**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M0.fvp_mps2.sse200.iotss_systemcontrol**

IoT Subsystem System Control registers.
Type: `IoTSS_SystemControl`.

**FVP_MPS2_Cortex_M0.fvp_mps2.sse200.iotss_systeminfo**

IoT Subsystem System Information registers.
Type: `IoTSS_SystemInfo`.

**FVP_MPS2_Cortex_M0.fvp_mps2.sse200.mem_switch_internal_msniper_tcm**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M0.fvp_mps2.sse200.mem_switch_internal_sram**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M0.fvp_mps2.sse200.mem_switch_internal_sram_mpc**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M0.fvp_mps2.sse200.mem_switch_ppu**

Allow transactions to be routed arbitrarily.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M0.fvp_mps2.sse200.mhu0**

IoT Subsystem Message Handling Unit.
Type: `IoTSS_MessageHandlingUnit`.

**FVP_MPS2_Cortex_M0.fvp_mps2.sse200.mhu1**

IoT Subsystem Message Handling Unit.
Type: `IoTSS_MessageHandlingUnit`.

**FVP_MPS2_Cortex_M0.fvp_mps2.sse200.mpc_iotss_internal_sram0**

IoT Subsystem Memory Protection Controller.
Type: `IoTSS_MemoryProtectionController`.

**FVP_MPS2_Cortex_M0.fvp_mps2.sse200.mpc_iotss_internal_sram0.gating_disabled_thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M0.fvp_mps2.sse200.mpc_iotss_internal_sram1**

IoT Subsystem Memory Protection Controller.
Type: `IoTSS_MemoryProtectionController`.

**FVP_MPS2_Cortex_M0.fvp_mps2.sse200.mpc_iotss_internal_sram1.gating_disabled_thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M0.fvp_mps2.sse200.mpc_iotss_internal_sram2**

IoT Subsystem Memory Protection Controller.
Type: `IoTSS_MemoryProtectionController`.

**FVP_MPS2_Cortex_M0.fvp_mps2.sse200.mpc_iotss_internal_sram2.gating_disabled_thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M0.fvp_mps2.sse200.mpc_iotss_internal_sram3**

IoT Subsystem Memory Protection Controller.
Type: `IoTSS_MemoryProtectionController`.

**FVP_MPS2_Cortex_M0.fvp_mps2.sse200.mpc_iotss_internal_sram3.gating_disabled_thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M0.fvp_mps2.sse200.nmi_or_gate**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_Cortex_M0.fvp_mps2.sse200.nonsecure_watchdog**

ARM Watchdog Module.
Type: `CMSDK_Watchdog`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M0.fvp_mps2.sse200.ram0_ppu**
ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M0.fvp_mps2.sse200.ram1_ppu**
ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M0.fvp_mps2.sse200.ram2_ppu**
ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M0.fvp_mps2.sse200.ram3_ppu**
ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M0.fvp_mps2.sse200.s32k_timer**
ARM Timer Module.
Type: `CMSDK_Timer`.

**FVP_MPS2_Cortex_M0.fvp_mps2.sse200.s32k_timer.clk_div**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M0.fvp_mps2.sse200.s32k_timer.counter**
Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M0.fvp_mps2.sse200.s32k_watchdog**
ARM Watchdog Module.
Type: `CMSDK_Watchdog`.

**FVP_MPS2_Cortex_M0.fvp_mps2.sse200.secure_control_register_block**
MPS2 Secure Control Register Block.
Type: `MPS2_SecureCtrl`.

**FVP_MPS2_Cortex_M0.fvp_mps2.sse200.secure_watchdog**
ARM Watchdog Module.
Type: `CMSDK_Watchdog`.

**FVP_MPS2_Cortex_M0.fvp_mps2.sse200.signal_router**
Signal router.
Type: `SignalRouter`.

**FVP_MPS2_Cortex_M0.fvp_mps2.sse200.sys_ppu**
ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M0.fvp_mps2.sse200.timer0**
ARM Timer Module.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `CMSDK_Timer`.

**FVP_MPS2_Cortex_M0.fvp_mps2.sse200.timer0.clk_div**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M0.fvp_mps2.sse200.timer0.counter**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M0.fvp_mps2.sse200.timer1**

ARM Timer Module.
Type: `CMSDK_Timer`.

**FVP_MPS2_Cortex_M0.fvp_mps2.sse200.timer1.clk_div**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M0.fvp_mps2.sse200.timer1.counter**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M0.fvp_mps2.ssram1**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M0.fvp_mps2.ssram2**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M0.fvp_mps2.stub0**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M0.fvp_mps2.stub1**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M0.fvp_mps2.stub_i2c1**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M0.fvp_mps2.stub_i2s**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M0.fvp_mps2.stub_spi0**

RAM device, can be dynamic or static ram.
Type: RAMDevice.

**FVP_MPS2_Cortex_M0.fvp_mps2.stub_spi2**

RAM device, can be dynamic or static ram.
Type: RAMDevice.

**FVP_MPS2_Cortex_M0.fvp_mps2.svos_dualtimer**

Type: SVOS_DualTimer.

**FVP_MPS2_Cortex_M0.fvp_mps2.svos_dualtimer.svos_dualtimer0**

ARM Dual-Timer Module.
Type: CMSDK_DualTimer.

**FVP_MPS2_Cortex_M0.fvp_mps2.svos_dualtimer.svos_dualtimer0.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: ClockDivider.

**FVP_MPS2_Cortex_M0.fvp_mps2.svos_dualtimer.svos_dualtimer0.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: ClockDivider.

**FVP_MPS2_Cortex_M0.fvp_mps2.svos_dualtimer.svos_dualtimer0.counter0**

Internal component used by SP804 Timer module.
Type: CounterModule.

**FVP_MPS2_Cortex_M0.fvp_mps2.svos_dualtimer.svos_dualtimer0.counter1**

Internal component used by SP804 Timer module.
Type: CounterModule.

**FVP_MPS2_Cortex_M0.fvp_mps2.svos_dualtimer.svos_dualtimer1**

ARM Dual-Timer Module.
Type: CMSDK_DualTimer.

**FVP_MPS2_Cortex_M0.fvp_mps2.svos_dualtimer.svos_dualtimer1.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: ClockDivider.

**FVP_MPS2_Cortex_M0.fvp_mps2.svos_dualtimer.svos_dualtimer1.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M0.fvp_mps2.svos_dualtimer.svos_dualtimer1.counter0**
Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M0.fvp_mps2.svos_dualtimer.svos_dualtimer1.counter1**
Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M0.fvp_mps2.svos_dualtimer.svos_dualtimer2**
ARM Dual-Timer Module.
Type: `CMSDK_DualTimer`.

**FVP_MPS2_Cortex_M0.fvp_mps2.svos_dualtimer.svos_dualtimer2.clk_div0**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M0.fvp_mps2.svos_dualtimer.svos_dualtimer2.clk_div1**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M0.fvp_mps2.svos_dualtimer.svos_dualtimer2.counter0**
Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M0.fvp_mps2.svos_dualtimer.svos_dualtimer2.counter1**
Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M0.fvp_mps2.svos_dualtimer.svos_dualtimer3**
ARM Dual-Timer Module.
Type: `CMSDK_DualTimer`.

**FVP_MPS2_Cortex_M0.fvp_mps2.svos_dualtimer.svos_dualtimer3.clk_div0**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M0.fvp_mps2.svos_dualtimer.svos_dualtimer3.clk_div1**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**`FVP_MPS2_Cortex_M0.fvp_mps2.svos_dualtimer.svos_dualtimer3.counter0`**
Internal component used by SP804 Timer module.
Type: `CounterModule`.

**`FVP_MPS2_Cortex_M0.fvp_mps2.svos_dualtimer.svos_dualtimer3.counter1`**
Internal component used by SP804 Timer module.
Type: `CounterModule`.

**`FVP_MPS2_Cortex_M0.fvp_mps2.switch_PSRAM_M7`**
Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**`FVP_MPS2_Cortex_M0.fvp_mps2.switch_svos_dualtimer`**
Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**`FVP_MPS2_Cortex_M0.fvp_mps2.telnetterminal0`**
Telnet terminal interface.
Type: `TelnetTerminal`.

**`FVP_MPS2_Cortex_M0.fvp_mps2.telnetterminal1`**
Telnet terminal interface.
Type: `TelnetTerminal`.

**`FVP_MPS2_Cortex_M0.fvp_mps2.telnetterminal2`**
Telnet terminal interface.
Type: `TelnetTerminal`.

**`FVP_MPS2_Cortex_M0.fvp_mps2.touchscreen_interface`**
MPS2 Touch Screen.
Type: `MPS2_TouchScreen`.

**`FVP_MPS2_Cortex_M0.fvp_mps2.uart_overflows_or_gate`**
Or Gate.
Type: `OrGate`.

## 16.3 FVP_MPS2_Cortex-M0plus

List of instances in FVP_MPS2_Cortex-M0plus.

### FVP_MPS2_Cortex-M0plus instances

**`FVP_MPS2_Cortex_M0plus`**
Type: `FVP_MPS2_Cortex_M0plus`.

**`FVP_MPS2_Cortex_M0plus.armcortexm0plusct`**
ARM CORTEXM0+ CT model.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ARM_Cortex-M0+`.

**FVP_MPS2_Cortex_M0plus.clk25Mhz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M0plus.clk25khz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2**

MPS2 DUT.
Type: `FVP_MPS2`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.GPIO0**

ARM PrimeCell General Purpose Input/Output.
Type: `CMSDK_GPIO`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.GPIO1**

ARM PrimeCell General Purpose Input/Output.
Type: `CMSDK_GPIO`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.GPIO2**

ARM PrimeCell General Purpose Input/Output.
Type: `CMSDK_GPIO`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.GPIO3**

ARM PrimeCell General Purpose Input/Output.
Type: `CMSDK_GPIO`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.GPIO_connection_test**

Type: `GPIO_Connection_Test`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.GPIO_connection_test.GPIO0_port_trans**

Type: `GPIO_Port_Transfer`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.GPIO_connection_test.GPIO1_port_test**

Type: `GPIO1_Connection_Test`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.PSRAM**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.PSRAM_M7**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M0plus.fvp_mps2.UART0**

ARM CMSDK UART Module.
Type: `CMSDK_UART`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.UART0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.UART1**

ARM CMSDK UART Module.
Type: `CMSDK_UART`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.UART1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.UART2**

ARM CMSDK UART Module.
Type: `CMSDK_UART`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.UART2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.VGA_interface**

VGA display interface between main bus and visualisation.
Type: `MPS2_VGA`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.ahb_ppc_iotss_expansion0**

IoT Subsystem Peripheral Protection Controller.
Type: `IoTSS_PeripheralProtectionController`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.ahb_ppc_iotss_expansion1**

IoT Subsystem Peripheral Protection Controller.
Type: `IoTSS_PeripheralProtectionController`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.apb_ppc_iotss_expansion0**

IoT Subsystem Peripheral Protection Controller.
Type: `IoTSS_PeripheralProtectionController`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.apb_ppc_iotss_expansion1**

IoT Subsystem Peripheral Protection Controller.
Type: `IoTSS_PeripheralProtectionController`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M0plus.fvp_mps2.apb_ppc_iotss_expansion2**

IoT Subsystem Peripheral Protection Controller.
Type: `IoTSS_PeripheralProtectionController`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.cmsdk_sysctrl**

Cortex-M Simple System Control.
Type: `CMSDK_SysCtrl`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.cmsdk_watchdog**

ARM Watchdog Module.
Type: `CMSDK_Watchdog`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.cpu_wait_or_gate_0**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.cpu_wait_or_gate_1**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.dbgen_or_gate**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.dma0**

ARM PrimeCell DMA Controller(PL080/081).
Type: `PL080_DMAC`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.dma0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M0plus.fvp_mps2.dma0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.dma0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.dma0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.dma0_idau_labeller**

Type: `LabellerIdauSecurity`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.dma0_securitymodifier**

Type: `SecurityModifier`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.dma1**

ARM PrimeCell DMA Controller(PL080/081).
Type: `PL080_DMAC`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.dma1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.dma1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.dma1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M0plus.fvp_mps2.dma1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.dma1_idau_labeller**

Type: `LabellerIdauSecurity`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.dma1_securitymodifier**

Type: `SecurityModifier`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.dma2**

ARM PrimeCell DMA Controller(PL080/081).
Type: `PL080_DMAC`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.dma2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.dma2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.dma2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.dma2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.dma2_idau_labeller**

Type: `LabellerIdauSecurity`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.dma2_securitymodifier**

Type: `SecurityModifier`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.dma3**

ARM PrimeCell DMA Controller(PL080/081).
Type: `PL080_DMAC`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M0plus.fvp_mps2.dma3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.dma3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.dma3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.dma3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.dma3_idau_labeller**

Type: `LabellerIdauSecurity`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.dma3_securitymodifier**

Type: `SecurityModifier`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.exclusive_monitor_psram**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.exclusive_monitor_psram_iotss**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.exclusive_monitor_switch_extra_psram_iotss**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.exclusive_monitor_switch_extra_psram_mps2**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.exclusive_monitor_zbtsram1**

Global exclusive monitor.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.exclusive_monitor_zbtsram2**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.exclusive_squasher**

Squashes the exclusive attribute on bus transactions.
Type: `PVBusExclusiveSquasher`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.fpga_sysctrl**

FPGA SysCtrl timers LEDs and switches.
Type: `FPGA_SysCtrl`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.fpga_sysctrl.callBack100HzCounter**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.fpga_sysctrl.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.gpio_0_or_2**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.gpio_1_or_3**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.mem_switch_extra_psram_iotss**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.mem_switch_extra_psram_mps2**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.mpc_iotss_ssram1**

IoT Subsystem Memory Protection Controller.
Type: `IoTSS_MemoryProtectionController`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.mpc_iotss_ssram1.gating_disabled_thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M0plus.fvp_mps2.mpc_iotss_ssram2**

IoT Subsystem Memory Protection Controller.
Type: `IoTSS_MemoryProtectionController`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.mpc_iotss_ssram2.gating_disabled_thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.mpc_iotss_ssram3**

IoT Subsystem Memory Protection Controller.
Type: `IoTSS_MemoryProtectionController`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.mpc_iotss_ssram3.gating_disabled_thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.mps2_audio**

MPS2 Audio.
Type: `MPS2_Audio`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.mps2_cmsdk_dualtimer**

ARM Dual-Timer Module.
Type: `CMSDK_DualTimer`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.mps2_cmsdk_dualtimer.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.mps2_cmsdk_dualtimer.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.mps2_cmsdk_dualtimer.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.mps2_cmsdk_dualtimer.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.mps2_exclusive_monitor_zbtsram1**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M0plus.fvp_mps2.mps2_exclusive_monitor_zbtsram2**

Global exclusive monitor.
Type: PVBusExclusiveMonitor.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.mps2_lcd**

MPS2 LCD I2C interface.
Type: MPS2_LCD.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.mps2_mpc_iotss_ssram1**

IoT Subsystem Memory Protection Controller.
Type: IoTSS_MemoryProtectionController.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.mps2_mpc_iotss_ssram1.gating_disabled_thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: SchedulerThreadEvent.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.mps2_secure_control_register_block**

MPS2 Secure Control Register Block.
Type: MPS2_SecureCtrl.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.mps2_timer0**

ARM Timer Module.
Type: CMSDK_Timer.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.mps2_timer0.clk_div**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: ClockDivider.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.mps2_timer0.counter**

Internal component used by SP804 Timer module.
Type: CounterModule.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.mps2_timer1**

ARM Timer Module.
Type: CMSDK_Timer.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.mps2_timer1.clk_div**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: ClockDivider.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.mps2_timer1.counter**

Internal component used by SP804 Timer module.
Type: CounterModule.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.mps2_visualisation**

Display window for MPS2 using Visualisation library.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `MPS2_Visualisation`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.niden_or_gate**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.nmi_or_gate**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.pl022_ssp_mps2**

ARM PrimeCell Synchronous Serial Port(PL022).
Type: `PL022_SSP_MPS2`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.pl022_ssp_mps2.prescaler**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.platform_bus_switch**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.platform_switch_dma0**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.platform_switch_dma1**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.signal_router**

Signal router.
Type: `SignalRouter`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.smsc_91c111**

SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.spiden_or_gate**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.spniden_or_gate**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.sse200**

SSE-200 subsystem.
Type: `SSE200`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M0plus.fvp_mps2.sse200.acg_cpu0**
IoT Subsystem Access Control Gate.
Type: `IoTSS_AccessControlGate`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.sse200.acg_cpu1**
IoT Subsystem Access Control Gate.
Type: `IoTSS_AccessControlGate`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.sse200.acg_sram0**
IoT Subsystem Access Control Gate.
Type: `IoTSS_AccessControlGate`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.sse200.acg_sram1**
IoT Subsystem Access Control Gate.
Type: `IoTSS_AccessControlGate`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.sse200.acg_sram2**
IoT Subsystem Access Control Gate.
Type: `IoTSS_AccessControlGate`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.sse200.acg_sram3**
IoT Subsystem Access Control Gate.
Type: `IoTSS_AccessControlGate`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.sse200.apb_ppc_iotss_subsystem0**
IoT Subsystem Peripheral Protection Controller.
Type: `IoTSS_PeripheralProtectionController`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.sse200.apb_ppc_iotss_subsystem1**
IoT Subsystem Peripheral Protection Controller.
Type: `IoTSS_PeripheralProtectionController`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.sse200.clock32kHz**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.sse200.clockdivider**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.sse200.cmsdk_dualtimer**
ARM Dual-Timer Module.
Type: `CMSDK_DualTimer`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.sse200.cmsdk_dualtimer.clk_div0**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.sse200.cmsdk_dualtimer.clk_div1**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.sse200.cmsdk_dualtimer.counter0**
Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.sse200.cmsdk_dualtimer.counter1**
Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.sse200.cordio_ppu**
ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.sse200.cpu0core_ppu**
ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.sse200.cpu0dbg_ppu**
ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.sse200.cpu1core_ppu**
ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.sse200.cpu1dbg_ppu**
ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.sse200.crypto_ppu**
ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.sse200.dbg_ppu**
ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.sse200.exclusive_monitor_iotss_internal_sram0**
Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.sse200.exclusive_monitor_iotss_internal_sram1**
Global exclusive monitor.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `PVBusExclusiveMonitor`.

**`FVP_MPS2_Cortex_M0plus.fvp_mps2.sse200.exclusive_monitor_iotss_internal_sram2`**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**`FVP_MPS2_Cortex_M0plus.fvp_mps2.sse200.exclusive_monitor_iotss_internal_sram3`**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**`FVP_MPS2_Cortex_M0plus.fvp_mps2.sse200.exclusive_squasher`**

Squashes the exclusive attribute on bus transactions.
Type: `PVBusExclusiveSquasher`.

**`FVP_MPS2_Cortex_M0plus.fvp_mps2.sse200.idau_labeller`**

Type: `LabellerIdauSecurity`.

**`FVP_MPS2_Cortex_M0plus.fvp_mps2.sse200.internal_msniper_tcm`**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**`FVP_MPS2_Cortex_M0plus.fvp_mps2.sse200.iotss_cpuidentity`**

IoT Subsystem CPU_IDENTITY registers.
Type: `IoTSS_CPUIdentity`.

**`FVP_MPS2_Cortex_M0plus.fvp_mps2.sse200.iotss_internal_sram0`**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**`FVP_MPS2_Cortex_M0plus.fvp_mps2.sse200.iotss_internal_sram1`**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**`FVP_MPS2_Cortex_M0plus.fvp_mps2.sse200.iotss_internal_sram2`**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**`FVP_MPS2_Cortex_M0plus.fvp_mps2.sse200.iotss_internal_sram3`**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**`FVP_MPS2_Cortex_M0plus.fvp_mps2.sse200.iotss_systemcontrol`**

IoT Subsystem System Control registers.
Type: `IoTSS_SystemControl`.

**`FVP_MPS2_Cortex_M0plus.fvp_mps2.sse200.iotss_systeminfo`**

IoT Subsystem System Information registers.
Type: `IoTSS_SystemInfo`.

**`FVP_MPS2_Cortex_M0plus.fvp_mps2.sse200.mem_switch_internal_msniper_tcm`**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M0plus.fvp_mps2.sse200.mem_switch_internal_sram**
Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.sse200.mem_switch_internal_sram_mpc**
Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.sse200.mem_switch_ppu**
Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.sse200.mhu0**
IoT Subsystem Message Handling Unit.
Type: `IoTSS_MessageHandlingUnit`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.sse200.mhu1**
IoT Subsystem Message Handling Unit.
Type: `IoTSS_MessageHandlingUnit`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.sse200.mpc_iotss_internal_sram0**
IoT Subsystem Memory Protection Controller.
Type: `IoTSS_MemoryProtectionController`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.sse200.mpc_iotss_internal_sram0.gating_disabled_thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.sse200.mpc_iotss_internal_sram1**
IoT Subsystem Memory Protection Controller.
Type: `IoTSS_MemoryProtectionController`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.sse200.mpc_iotss_internal_sram1.gating_disabled_thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.sse200.mpc_iotss_internal_sram2**
IoT Subsystem Memory Protection Controller.
Type: `IoTSS_MemoryProtectionController`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.sse200.mpc_iotss_internal_sram2.gating_disabled_thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.sse200.mpc_iotss_internal_sram3**
IoT Subsystem Memory Protection Controller.
Type: `IoTSS_MemoryProtectionController`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M0plus.fvp_mps2.sse200.mpc_iotss_internal_sram3.gating_disabled_thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.sse200.nmi_or_gate**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.sse200.nonsecure_watchdog**

ARM Watchdog Module.
Type: `CMSDK_Watchdog`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.sse200.ram0_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.sse200.ram1_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.sse200.ram2_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.sse200.ram3_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.sse200.s32k_timer**

ARM Timer Module.
Type: `CMSDK_Timer`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.sse200.s32k_timer.clk_div**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.sse200.s32k_timer.counter**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.sse200.s32k_watchdog**

ARM Watchdog Module.
Type: `CMSDK_Watchdog`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.sse200.secure_control_register_block**

MPS2 Secure Control Register Block.
Type: `MPS2_SecureCtrl`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M0plus.fvp_mps2.sse200.secure_watchdog**
ARM Watchdog Module.
Type: `CMSDK_Watchdog`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.sse200.signal_router**
Signal router.
Type: `SignalRouter`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.sse200.sys_ppu**
ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.sse200.timer0**
ARM Timer Module.
Type: `CMSDK_Timer`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.sse200.timer0.clk_div**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.sse200.timer0.counter**
Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.sse200.timer1**
ARM Timer Module.
Type: `CMSDK_Timer`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.sse200.timer1.clk_div**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.sse200.timer1.counter**
Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.ssram1**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.ssram2**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.stub0**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M0plus.fvp_mps2.stub1**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.stub_i2c1**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.stub_i2s**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.stub_spi0**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.stub_spi2**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.svos_dualtimer**

Type: `SVOS_DualTimer`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.svos_dualtimer.svos_dualtimer0**

ARM Dual-Timer Module.
Type: `CMSDK_DualTimer`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.svos_dualtimer.svos_dualtimer0.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.svos_dualtimer.svos_dualtimer0.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.svos_dualtimer.svos_dualtimer0.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.svos_dualtimer.svos_dualtimer0.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.svos_dualtimer.svos_dualtimer1**

ARM Dual-Timer Module.
Type: `CMSDK_DualTimer`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M0plus.fvp_mps2.svos_dualtimer.svos_dualtimer1.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.svos_dualtimer.svos_dualtimer1.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.svos_dualtimer.svos_dualtimer1.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.svos_dualtimer.svos_dualtimer1.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.svos_dualtimer.svos_dualtimer2**

ARM Dual-Timer Module.
Type: `CMSDK_DualTimer`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.svos_dualtimer.svos_dualtimer2.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.svos_dualtimer.svos_dualtimer2.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.svos_dualtimer.svos_dualtimer2.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.svos_dualtimer.svos_dualtimer2.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.svos_dualtimer.svos_dualtimer3**

ARM Dual-Timer Module.
Type: `CMSDK_DualTimer`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M0plus.fvp_mps2.svos_dualtimer.svos_dualtimer3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.svos_dualtimer.svos_dualtimer3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.svos_dualtimer.svos_dualtimer3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.svos_dualtimer.svos_dualtimer3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.switch_PSRAM_M7**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.switch_svos_dualtimer**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.telnetterminal0**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.telnetterminal1**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.telnetterminal2**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.touchscreen_interface**

MPS2 Touch Screen.
Type: `MPS2_TouchScreen`.

**FVP_MPS2_Cortex_M0plus.fvp_mps2.uart_overflows_or_gate**

Or Gate.
Type: `OrGate`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

# 16.4 FVP_MPS2_Cortex-M23

List of instances in FVP_MPS2_Cortex-M23.

## FVP_MPS2_Cortex-M23 instances

**FVP_MPS2_Cortex_M23**

Type: `FVP_MPS2_Cortex_M23`.

**FVP_MPS2_Cortex_M23.clk25Mhz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M23.clk25khz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M23.cpu0**

ARM CORTEXM23 CT model.
Type: `ARM_Cortex-M23`.

**FVP_MPS2_Cortex_M23.cpu1**

ARM CORTEXM23 CT model.
Type: `ARM_Cortex-M23`.

**FVP_MPS2_Cortex_M23.fvp_mps2**

MPS2 DUT.
Type: `FVP_MPS2`.

**FVP_MPS2_Cortex_M23.fvp_mps2.GPIO0**

ARM PrimeCell General Purpose Input/Output.
Type: `CMSDK_GPIO`.

**FVP_MPS2_Cortex_M23.fvp_mps2.GPIO1**

ARM PrimeCell General Purpose Input/Output.
Type: `CMSDK_GPIO`.

**FVP_MPS2_Cortex_M23.fvp_mps2.GPIO2**

ARM PrimeCell General Purpose Input/Output.
Type: `CMSDK_GPIO`.

**FVP_MPS2_Cortex_M23.fvp_mps2.GPIO3**

ARM PrimeCell General Purpose Input/Output.
Type: `CMSDK_GPIO`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M23.fvp_mps2.GPIO_connection_test**

Type: `GPIO_Connection_Test`.

**FVP_MPS2_Cortex_M23.fvp_mps2.GPIO_connection_test.GPIO0_port_trans**

Type: `GPIO_Port_Transfer`.

**FVP_MPS2_Cortex_M23.fvp_mps2.GPIO_connection_test.GPIO1_port_test**

Type: `GPIO1_Connection_Test`.

**FVP_MPS2_Cortex_M23.fvp_mps2.PSRAM**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M23.fvp_mps2.PSRAM_M7**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M23.fvp_mps2.UART0**

ARM CMSDK UART Module.
Type: `CMSDK_UART`.

**FVP_MPS2_Cortex_M23.fvp_mps2.UART0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M23.fvp_mps2.UART1**

ARM CMSDK UART Module.
Type: `CMSDK_UART`.

**FVP_MPS2_Cortex_M23.fvp_mps2.UART1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M23.fvp_mps2.UART2**

ARM CMSDK UART Module.
Type: `CMSDK_UART`.

**FVP_MPS2_Cortex_M23.fvp_mps2.UART2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M23.fvp_mps2.VGA_interface**

VGA display interface between main bus and visualisation.
Type: `MPS2_VGA`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M23.fvp_mps2.ahb_ppc_iotss_expansion0**

IoT Subsystem Peripheral Protection Controller.
Type: `IoTSS_PeripheralProtectionController`.

**FVP_MPS2_Cortex_M23.fvp_mps2.ahb_ppc_iotss_expansion1**

IoT Subsystem Peripheral Protection Controller.
Type: `IoTSS_PeripheralProtectionController`.

**FVP_MPS2_Cortex_M23.fvp_mps2.apb_ppc_iotss_expansion0**

IoT Subsystem Peripheral Protection Controller.
Type: `IoTSS_PeripheralProtectionController`.

**FVP_MPS2_Cortex_M23.fvp_mps2.apb_ppc_iotss_expansion1**

IoT Subsystem Peripheral Protection Controller.
Type: `IoTSS_PeripheralProtectionController`.

**FVP_MPS2_Cortex_M23.fvp_mps2.apb_ppc_iotss_expansion2**

IoT Subsystem Peripheral Protection Controller.
Type: `IoTSS_PeripheralProtectionController`.

**FVP_MPS2_Cortex_M23.fvp_mps2.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M23.fvp_mps2.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M23.fvp_mps2.cmsdk_sysctrl**

Cortex-M Simple System Control.
Type: `CMSDK_SysCtrl`.

**FVP_MPS2_Cortex_M23.fvp_mps2.cmsdk_watchdog**

ARM Watchdog Module.
Type: `CMSDK_Watchdog`.

**FVP_MPS2_Cortex_M23.fvp_mps2.cpu_wait_or_gate_0**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_Cortex_M23.fvp_mps2.cpu_wait_or_gate_1**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_Cortex_M23.fvp_mps2.dbgen_or_gate**

Or Gate.
Type: `OrGate`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M23.fvp_mps2.dma0**

ARM PrimeCell DMA Controller(PL080/081).
Type: `PL080_DMAC`.

**FVP_MPS2_Cortex_M23.fvp_mps2.dma0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_MPS2_Cortex_M23.fvp_mps2.dma0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_MPS2_Cortex_M23.fvp_mps2.dma0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_MPS2_Cortex_M23.fvp_mps2.dma0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M23.fvp_mps2.dma0_idau_labeller**

Type: `LabellerIdauSecurity`.

**FVP_MPS2_Cortex_M23.fvp_mps2.dma0_securitymodifier**

Type: `SecurityModifier`.

**FVP_MPS2_Cortex_M23.fvp_mps2.dma1**

ARM PrimeCell DMA Controller(PL080/081).
Type: `PL080_DMAC`.

**FVP_MPS2_Cortex_M23.fvp_mps2.dma1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M23.fvp_mps2.dma1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_MPS2_Cortex_M23.fvp_mps2.dma1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_MPS2_Cortex_M23.fvp_mps2.dma1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M23.fvp_mps2.dma1_idau_labeller**

Type: `LabellerIdauSecurity`.

**FVP_MPS2_Cortex_M23.fvp_mps2.dma1_securitymodifier**

Type: `SecurityModifier`.

**FVP_MPS2_Cortex_M23.fvp_mps2.dma2**

ARM PrimeCell DMA Controller(PL080/081).
Type: `PL080_DMAC`.

**FVP_MPS2_Cortex_M23.fvp_mps2.dma2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_MPS2_Cortex_M23.fvp_mps2.dma2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_MPS2_Cortex_M23.fvp_mps2.dma2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M23.fvp_mps2.dma2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M23.fvp_mps2.dma2_idau_labeller**

Type: `LabellerIdauSecurity`.

**FVP_MPS2_Cortex_M23.fvp_mps2.dma2_securitymodifier**

Type: `SecurityModifier`.

**FVP_MPS2_Cortex_M23.fvp_mps2.dma3**

ARM PrimeCell DMA Controller(PL080/081).
Type: `PL080_DMAC`.

**FVP_MPS2_Cortex_M23.fvp_mps2.dma3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_MPS2_Cortex_M23.fvp_mps2.dma3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_MPS2_Cortex_M23.fvp_mps2.dma3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_MPS2_Cortex_M23.fvp_mps2.dma3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M23.fvp_mps2.dma3_idau_labeller**

Type: `LabellerIdauSecurity`.

**FVP_MPS2_Cortex_M23.fvp_mps2.dma3_securitymodifier**

Type: `SecurityModifier`.

**FVP_MPS2_Cortex_M23.fvp_mps2.exclusive_monitor_psram**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M23.fvp_mps2.exclusive_monitor_psram_iotss**
Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_Cortex_M23.fvp_mps2.exclusive_monitor_switch_extra_psram_iotss**
Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M23.fvp_mps2.exclusive_monitor_switch_extra_psram_mps2**
Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M23.fvp_mps2.exclusive_monitor_zbtsram1**
Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_Cortex_M23.fvp_mps2.exclusive_monitor_zbtsram2**
Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_Cortex_M23.fvp_mps2.exclusive_squasher**
Squashes the exclusive attribute on bus transactions.
Type: `PVBusExclusiveSquasher`.

**FVP_MPS2_Cortex_M23.fvp_mps2.fpga_sysctrl**
FPGA SysCtrl timers LEDs and switches.
Type: `FPGA_SysCtrl`.

**FVP_MPS2_Cortex_M23.fvp_mps2.fpga_sysctrl.callBack100HzCounter**
Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M23.fvp_mps2.fpga_sysctrl.clockdivider**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M23.fvp_mps2.gpio_0_or_2**
Or Gate.
Type: `OrGate`.

**FVP_MPS2_Cortex_M23.fvp_mps2.gpio_1_or_3**
Or Gate.
Type: `OrGate`.

**FVP_MPS2_Cortex_M23.fvp_mps2.hostbridge**
Host Socket Interface Component.
Type: `HostBridge`.

**FVP_MPS2_Cortex_M23.fvp_mps2.mem_switch_extra_psram_iotss**
Allow transactions to be routed arbitrarily.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M23.fvp_mps2.mem_switch_extra_psram_mps2**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M23.fvp_mps2.mpc_iotss_ssram1**

IoT Subsystem Memory Protection Controller.
Type: `IoTSS_MemoryProtectionController`.

**FVP_MPS2_Cortex_M23.fvp_mps2.mpc_iotss_ssram1.gating_disabled_thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M23.fvp_mps2.mpc_iotss_ssram2**

IoT Subsystem Memory Protection Controller.
Type: `IoTSS_MemoryProtectionController`.

**FVP_MPS2_Cortex_M23.fvp_mps2.mpc_iotss_ssram2.gating_disabled_thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M23.fvp_mps2.mpc_iotss_ssram3**

IoT Subsystem Memory Protection Controller.
Type: `IoTSS_MemoryProtectionController`.

**FVP_MPS2_Cortex_M23.fvp_mps2.mpc_iotss_ssram3.gating_disabled_thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M23.fvp_mps2.mps2_audio**

MPS2 Audio.
Type: `MPS2_Audio`.

**FVP_MPS2_Cortex_M23.fvp_mps2.mps2_cmsdk_dualtimer**

ARM Dual-Timer Module.
Type: `CMSDK_DualTimer`.

**FVP_MPS2_Cortex_M23.fvp_mps2.mps2_cmsdk_dualtimer.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M23.fvp_mps2.mps2_cmsdk_dualtimer.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ClockDivider`.

**FVP_MPS2_Cortex_M23.fvp_mps2.mps2_cmsdk_dualtimer.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M23.fvp_mps2.mps2_cmsdk_dualtimer.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M23.fvp_mps2.mps2_exclusive_monitor_zbtsram1**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_Cortex_M23.fvp_mps2.mps2_exclusive_monitor_zbtsram2**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_Cortex_M23.fvp_mps2.mps2_lcd**

MPS2 LCD I2C interface.
Type: `MPS2_LCD`.

**FVP_MPS2_Cortex_M23.fvp_mps2.mps2_mpc_iotss_ssram1**

IoT Subsystem Memory Protection Controller.
Type: `IoTSS_MemoryProtectionController`.

**FVP_MPS2_Cortex_M23.fvp_mps2.mps2_mpc_iotss_ssram1.gating_disabled_thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M23.fvp_mps2.mps2_secure_control_register_block**

MPS2 Secure Control Register Block.
Type: `MPS2_SecureCtrl`.

**FVP_MPS2_Cortex_M23.fvp_mps2.mps2_timer0**

ARM Timer Module.
Type: `CMSDK_Timer`.

**FVP_MPS2_Cortex_M23.fvp_mps2.mps2_timer0.clk_div**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M23.fvp_mps2.mps2_timer0.counter**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M23.fvp_mps2.mps2_timer1**

ARM Timer Module.
Type: `CMSDK_Timer`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M23.fvp_mps2.mps2_timer1.clk_div**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M23.fvp_mps2.mps2_timer1.counter**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M23.fvp_mps2.mps2_visualisation**

Display window for MPS2 using Visualisation library.
Type: `MPS2_Visualisation`.

**FVP_MPS2_Cortex_M23.fvp_mps2.niden_or_gate**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_Cortex_M23.fvp_mps2.nmi_or_gate**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_Cortex_M23.fvp_mps2.pl022_ssp_mps2**

ARM PrimeCell Synchronous Serial Port(PL022).
Type: `PL022_SSP_MPS2`.

**FVP_MPS2_Cortex_M23.fvp_mps2.pl022_ssp_mps2.prescaler**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M23.fvp_mps2.platform_bus_switch**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M23.fvp_mps2.platform_switch_dma0**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M23.fvp_mps2.platform_switch_dma1**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M23.fvp_mps2.signal_router**

Signal router.
Type: `SignalRouter`.

**FVP_MPS2_Cortex_M23.fvp_mps2.smsc_91c111**

SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M23.fvp_mps2.spiden_or_gate**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_Cortex_M23.fvp_mps2.spniden_or_gate**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_Cortex_M23.fvp_mps2.sse200**

SSE-200 subsystem.
Type: `SSE200`.

**FVP_MPS2_Cortex_M23.fvp_mps2.sse200.acg_cpu0**

IoT Subsystem Access Control Gate.
Type: `IoTSS_AccessControlGate`.

**FVP_MPS2_Cortex_M23.fvp_mps2.sse200.acg_cpu1**

IoT Subsystem Access Control Gate.
Type: `IoTSS_AccessControlGate`.

**FVP_MPS2_Cortex_M23.fvp_mps2.sse200.acg_sram0**

IoT Subsystem Access Control Gate.
Type: `IoTSS_AccessControlGate`.

**FVP_MPS2_Cortex_M23.fvp_mps2.sse200.acg_sram1**

IoT Subsystem Access Control Gate.
Type: `IoTSS_AccessControlGate`.

**FVP_MPS2_Cortex_M23.fvp_mps2.sse200.acg_sram2**

IoT Subsystem Access Control Gate.
Type: `IoTSS_AccessControlGate`.

**FVP_MPS2_Cortex_M23.fvp_mps2.sse200.acg_sram3**

IoT Subsystem Access Control Gate.
Type: `IoTSS_AccessControlGate`.

**FVP_MPS2_Cortex_M23.fvp_mps2.sse200.apb_ppc_iotss_subsystem0**

IoT Subsystem Peripheral Protection Controller.
Type: `IoTSS_PeripheralProtectionController`.

**FVP_MPS2_Cortex_M23.fvp_mps2.sse200.apb_ppc_iotss_subsystem1**

IoT Subsystem Peripheral Protection Controller.
Type: `IoTSS_PeripheralProtectionController`.

**FVP_MPS2_Cortex_M23.fvp_mps2.sse200.clock32kHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M23.fvp_mps2.sse200.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M23.fvp_mps2.sse200.cmsdk_dualtimer**

ARM Dual-Timer Module.
Type: `CMSDK_DualTimer`.

**FVP_MPS2_Cortex_M23.fvp_mps2.sse200.cmsdk_dualtimer.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M23.fvp_mps2.sse200.cmsdk_dualtimer.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M23.fvp_mps2.sse200.cmsdk_dualtimer.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M23.fvp_mps2.sse200.cmsdk_dualtimer.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M23.fvp_mps2.sse200.cordio_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M23.fvp_mps2.sse200.cpu0core_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M23.fvp_mps2.sse200.cpu0dbg_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M23.fvp_mps2.sse200.cpu1core_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M23.fvp_mps2.sse200.cpu1dbg_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M23.fvp_mps2.sse200.crypto_ppu**
ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M23.fvp_mps2.sse200.dbg_ppu**
ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M23.fvp_mps2.sse200.exclusive_monitor_iotss_internal_sram0**
Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_Cortex_M23.fvp_mps2.sse200.exclusive_monitor_iotss_internal_sram1**
Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_Cortex_M23.fvp_mps2.sse200.exclusive_monitor_iotss_internal_sram2**
Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_Cortex_M23.fvp_mps2.sse200.exclusive_monitor_iotss_internal_sram3**
Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_Cortex_M23.fvp_mps2.sse200.exclusive_squasher**
Squashes the exclusive attribute on bus transactions.
Type: `PVBusExclusiveSquasher`.

**FVP_MPS2_Cortex_M23.fvp_mps2.sse200.idau_labeller**
Type: `LabellerIdauSecurity`.

**FVP_MPS2_Cortex_M23.fvp_mps2.sse200.internal_msniper_tcm**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M23.fvp_mps2.sse200.iotss_cpuidentity**
IoT Subsystem CPU_IDENTITY registers.
Type: `IoTSS_CPUIdentity`.

**FVP_MPS2_Cortex_M23.fvp_mps2.sse200.iotss_internal_sram0**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M23.fvp_mps2.sse200.iotss_internal_sram1**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M23.fvp_mps2.sse200.iotss_internal_sram2**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M23.fvp_mps2.sse200.iotss_internal_sram3**
RAM device, can be dynamic or static ram.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `RAMDevice`.

**FVP_MPS2_Cortex_M23.fvp_mps2.sse200.iotss_systemcontrol**

IoT Subsystem System Control registers.
Type: `IoTSS_SystemControl`.

**FVP_MPS2_Cortex_M23.fvp_mps2.sse200.iotss_systeminfo**

IoT Subsystem System Information registers.
Type: `IoTSS_SystemInfo`.

**FVP_MPS2_Cortex_M23.fvp_mps2.sse200.mem_switch_internal_msniper_tcm**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M23.fvp_mps2.sse200.mem_switch_internal_sram**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M23.fvp_mps2.sse200.mem_switch_internal_sram_mpc**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M23.fvp_mps2.sse200.mem_switch_ppu**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M23.fvp_mps2.sse200.mhu0**

IoT Subsystem Message Handling Unit.
Type: `IoTSS_MessageHandlingUnit`.

**FVP_MPS2_Cortex_M23.fvp_mps2.sse200.mhu1**

IoT Subsystem Message Handling Unit.
Type: `IoTSS_MessageHandlingUnit`.

**FVP_MPS2_Cortex_M23.fvp_mps2.sse200.mpc_iotss_internal_sram0**

IoT Subsystem Memory Protection Controller.
Type: `IoTSS_MemoryProtectionController`.

**FVP_MPS2_Cortex_M23.fvp_mps2.sse200.mpc_iotss_internal_sram0.gating_disabled_thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M23.fvp_mps2.sse200.mpc_iotss_internal_sram1**

IoT Subsystem Memory Protection Controller.
Type: `IoTSS_MemoryProtectionController`.

**FVP_MPS2_Cortex_M23.fvp_mps2.sse200.mpc_iotss_internal_sram1.gating_disabled_thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M23.fvp_mps2.sse200.mpc_iotss_internal_sram2**

IoT Subsystem Memory Protection Controller.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `IoTSS_MemoryProtectionController`.

**FVP_MPS2_Cortex_M23.fvp_mps2.sse200.mpc_iotss_internal_sram2.gating_disabled_thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M23.fvp_mps2.sse200.mpc_iotss_internal_sram3**

IoT Subsystem Memory Protection Controller.
Type: `IoTSS_MemoryProtectionController`.

**FVP_MPS2_Cortex_M23.fvp_mps2.sse200.mpc_iotss_internal_sram3.gating_disabled_thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M23.fvp_mps2.sse200.nmi_or_gate**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_Cortex_M23.fvp_mps2.sse200.nonsecure_watchdog**

ARM Watchdog Module.
Type: `CMSDK_Watchdog`.

**FVP_MPS2_Cortex_M23.fvp_mps2.sse200.ram0_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M23.fvp_mps2.sse200.ram1_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M23.fvp_mps2.sse200.ram2_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M23.fvp_mps2.sse200.ram3_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M23.fvp_mps2.sse200.s32k_timer**

ARM Timer Module.
Type: `CMSDK_Timer`.

**FVP_MPS2_Cortex_M23.fvp_mps2.sse200.s32k_timer.clk_div**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M23.fvp_mps2.sse200.s32k_timer.counter**

Internal component used by SP804 Timer module.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `CounterModule`.

**FVP_MPS2_Cortex_M23.fvp_mps2.sse200.s32k_watchdog**

ARM Watchdog Module.
Type: `CMSDK_Watchdog`.

**FVP_MPS2_Cortex_M23.fvp_mps2.sse200.secure_control_register_block**

MPS2 Secure Control Register Block.
Type: `MPS2_SecureCtrl`.

**FVP_MPS2_Cortex_M23.fvp_mps2.sse200.secure_watchdog**

ARM Watchdog Module.
Type: `CMSDK_Watchdog`.

**FVP_MPS2_Cortex_M23.fvp_mps2.sse200.signal_router**

Signal router.
Type: `SignalRouter`.

**FVP_MPS2_Cortex_M23.fvp_mps2.sse200.sys_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M23.fvp_mps2.sse200.timer0**

ARM Timer Module.
Type: `CMSDK_Timer`.

**FVP_MPS2_Cortex_M23.fvp_mps2.sse200.timer0.clk_div**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M23.fvp_mps2.sse200.timer0.counter**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M23.fvp_mps2.sse200.timer1**

ARM Timer Module.
Type: `CMSDK_Timer`.

**FVP_MPS2_Cortex_M23.fvp_mps2.sse200.timer1.clk_div**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M23.fvp_mps2.sse200.timer1.counter**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M23.fvp_mps2.ssram1**
RAM device, can be dynamic or static ram.
Type: RAMDevice.

**FVP_MPS2_Cortex_M23.fvp_mps2.ssram2**
RAM device, can be dynamic or static ram.
Type: RAMDevice.

**FVP_MPS2_Cortex_M23.fvp_mps2.stub0**
RAM device, can be dynamic or static ram.
Type: RAMDevice.

**FVP_MPS2_Cortex_M23.fvp_mps2.stub1**
RAM device, can be dynamic or static ram.
Type: RAMDevice.

**FVP_MPS2_Cortex_M23.fvp_mps2.stub_i2c1**
RAM device, can be dynamic or static ram.
Type: RAMDevice.

**FVP_MPS2_Cortex_M23.fvp_mps2.stub_i2s**
RAM device, can be dynamic or static ram.
Type: RAMDevice.

**FVP_MPS2_Cortex_M23.fvp_mps2.stub_spi0**
RAM device, can be dynamic or static ram.
Type: RAMDevice.

**FVP_MPS2_Cortex_M23.fvp_mps2.stub_spi2**
RAM device, can be dynamic or static ram.
Type: RAMDevice.

**FVP_MPS2_Cortex_M23.fvp_mps2.svos_dualtimer**
Type: SVOS_DualTimer.

**FVP_MPS2_Cortex_M23.fvp_mps2.svos_dualtimer.svos_dualtimer0**
ARM Dual-Timer Module.
Type: CMSDK_DualTimer.

**FVP_MPS2_Cortex_M23.fvp_mps2.svos_dualtimer.svos_dualtimer0.clk_div0**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: ClockDivider.

**FVP_MPS2_Cortex_M23.fvp_mps2.svos_dualtimer.svos_dualtimer0.clk_div1**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: ClockDivider.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M23.fvp_mps2.svos_dualtimer.svos_dualtimer0.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M23.fvp_mps2.svos_dualtimer.svos_dualtimer0.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M23.fvp_mps2.svos_dualtimer.svos_dualtimer1**

ARM Dual-Timer Module.
Type: `CMSDK_DualTimer`.

**FVP_MPS2_Cortex_M23.fvp_mps2.svos_dualtimer.svos_dualtimer1.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M23.fvp_mps2.svos_dualtimer.svos_dualtimer1.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M23.fvp_mps2.svos_dualtimer.svos_dualtimer1.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M23.fvp_mps2.svos_dualtimer.svos_dualtimer1.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M23.fvp_mps2.svos_dualtimer.svos_dualtimer2**

ARM Dual-Timer Module.
Type: `CMSDK_DualTimer`.

**FVP_MPS2_Cortex_M23.fvp_mps2.svos_dualtimer.svos_dualtimer2.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M23.fvp_mps2.svos_dualtimer.svos_dualtimer2.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M23.fvp_mps2.svos_dualtimer.svos_dualtimer2.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M23.fvp_mps2.svos_dualtimer.svos_dualtimer2.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M23.fvp_mps2.svos_dualtimer.svos_dualtimer3**

ARM Dual-Timer Module.
Type: `CMSDK_DualTimer`.

**FVP_MPS2_Cortex_M23.fvp_mps2.svos_dualtimer.svos_dualtimer3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M23.fvp_mps2.svos_dualtimer.svos_dualtimer3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M23.fvp_mps2.svos_dualtimer.svos_dualtimer3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M23.fvp_mps2.svos_dualtimer.svos_dualtimer3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M23.fvp_mps2.switch_PSRAM_M7**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M23.fvp_mps2.switch_svos_dualtimer**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M23.fvp_mps2.telnetterminal0**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_MPS2_Cortex_M23.fvp_mps2.telnetterminal1**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_MPS2_Cortex_M23.fvp_mps2.telnetterminal2**

Telnet terminal interface.
Type: `TelnetTerminal`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M23.fvp_mps2.touchscreen_interface**

MPS2 Touch Screen.
Type: `MPS2_TouchScreen`.

**FVP_MPS2_Cortex_M23.fvp_mps2.uart_overflows_or_gate**

Or Gate.
Type: `OrGate`.

## 16.5 FVP_MPS2_Cortex-M3

List of instances in FVP_MPS2_Cortex-M3.

### FVP_MPS2_Cortex-M3 instances

**FVP_MPS2_Cortex_M3**

Type: `FVP_MPS2_Cortex_M3`.

**FVP_MPS2_Cortex_M3.armcortexm3ct**

ARM CORTEXM3 CT model.
Type: `ARM_Cortex-M3`.

**FVP_MPS2_Cortex_M3.clk25Mhz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M3.clk25khz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M3.fvp_mps2**

MPS2 DUT.
Type: `FVP_MPS2`.

**FVP_MPS2_Cortex_M3.fvp_mps2.GPIO0**

ARM PrimeCell General Purpose Input/Output.
Type: `CMSDK_GPIO`.

**FVP_MPS2_Cortex_M3.fvp_mps2.GPIO1**

ARM PrimeCell General Purpose Input/Output.
Type: `CMSDK_GPIO`.

**FVP_MPS2_Cortex_M3.fvp_mps2.GPIO2**

ARM PrimeCell General Purpose Input/Output.
Type: `CMSDK_GPIO`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M3.fvp_mps2.GPIO3**

ARM PrimeCell General Purpose Input/Output.
Type: `CMSDK_GPIO`.

**FVP_MPS2_Cortex_M3.fvp_mps2.GPIO_connection_test**

Type: `GPIO_Connection_Test`.

**FVP_MPS2_Cortex_M3.fvp_mps2.GPIO_connection_test.GPIO0_port_trans**

Type: `GPIO_Port_Transfer`.

**FVP_MPS2_Cortex_M3.fvp_mps2.GPIO_connection_test.GPIO1_port_test**

Type: `GPIO1_Connection_Test`.

**FVP_MPS2_Cortex_M3.fvp_mps2.PSRAM**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M3.fvp_mps2.PSRAM_M7**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M3.fvp_mps2.UART0**

ARM CMSDK UART Module.
Type: `CMSDK_UART`.

**FVP_MPS2_Cortex_M3.fvp_mps2.UART0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M3.fvp_mps2.UART1**

ARM CMSDK UART Module.
Type: `CMSDK_UART`.

**FVP_MPS2_Cortex_M3.fvp_mps2.UART1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M3.fvp_mps2.UART2**

ARM CMSDK UART Module.
Type: `CMSDK_UART`.

**FVP_MPS2_Cortex_M3.fvp_mps2.UART2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M3.fvp_mps2.VGA_interface**

VGA display interface between main bus and visualisation.
Type: `MPS2_VGA`.

**FVP_MPS2_Cortex_M3.fvp_mps2.ahb_ppc_iotss_expansion0**

IoT Subsystem Peripheral Protection Controller.
Type: `IoTSS_PeripheralProtectionController`.

**FVP_MPS2_Cortex_M3.fvp_mps2.ahb_ppc_iotss_expansion1**

IoT Subsystem Peripheral Protection Controller.
Type: `IoTSS_PeripheralProtectionController`.

**FVP_MPS2_Cortex_M3.fvp_mps2.apb_ppc_iotss_expansion0**

IoT Subsystem Peripheral Protection Controller.
Type: `IoTSS_PeripheralProtectionController`.

**FVP_MPS2_Cortex_M3.fvp_mps2.apb_ppc_iotss_expansion1**

IoT Subsystem Peripheral Protection Controller.
Type: `IoTSS_PeripheralProtectionController`.

**FVP_MPS2_Cortex_M3.fvp_mps2.apb_ppc_iotss_expansion2**

IoT Subsystem Peripheral Protection Controller.
Type: `IoTSS_PeripheralProtectionController`.

**FVP_MPS2_Cortex_M3.fvp_mps2.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M3.fvp_mps2.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M3.fvp_mps2.cmsdk_sysctrl**

Cortex-M Simple System Control.
Type: `CMSDK_SysCtrl`.

**FVP_MPS2_Cortex_M3.fvp_mps2.cmsdk_watchdog**

ARM Watchdog Module.
Type: `CMSDK_Watchdog`.

**FVP_MPS2_Cortex_M3.fvp_mps2.cpu_wait_or_gate_0**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_Cortex_M3.fvp_mps2.cpu_wait_or_gate_1**

Or Gate.
Type: `OrGate`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M3.fvp_mps2.dbgen_or_gate**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_Cortex_M3.fvp_mps2.dma0**

ARM PrimeCell DMA Controller(PL080/081).
Type: `PL080_DMAC`.

**FVP_MPS2_Cortex_M3.fvp_mps2.dma0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_MPS2_Cortex_M3.fvp_mps2.dma0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_MPS2_Cortex_M3.fvp_mps2.dma0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_MPS2_Cortex_M3.fvp_mps2.dma0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M3.fvp_mps2.dma0_idau_labeller**

Type: `LabellerIdauSecurity`.

**FVP_MPS2_Cortex_M3.fvp_mps2.dma0_securitymodifier**

Type: `SecurityModifier`.

**FVP_MPS2_Cortex_M3.fvp_mps2.dma1**

ARM PrimeCell DMA Controller(PL080/081).
Type: `PL080_DMAC`.

**FVP_MPS2_Cortex_M3.fvp_mps2.dma1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_MPS2_Cortex_M3.fvp_mps2.dma1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_MPS2_Cortex_M3.fvp_mps2.dma1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_MPS2_Cortex_M3.fvp_mps2.dma1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M3.fvp_mps2.dma1_idau_labeller**

Type: `LabellerIdauSecurity`.

**FVP_MPS2_Cortex_M3.fvp_mps2.dma1_securitymodifier**

Type: `SecurityModifier`.

**FVP_MPS2_Cortex_M3.fvp_mps2.dma2**

ARM PrimeCell DMA Controller(PL080/081).
Type: `PL080_DMAC`.

**FVP_MPS2_Cortex_M3.fvp_mps2.dma2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_MPS2_Cortex_M3.fvp_mps2.dma2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M3.fvp_mps2.dma2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_MPS2_Cortex_M3.fvp_mps2.dma2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M3.fvp_mps2.dma2_idau_labeller**

Type: `LabellerIdauSecurity`.

**FVP_MPS2_Cortex_M3.fvp_mps2.dma2_securitymodifier**

Type: `SecurityModifier`.

**FVP_MPS2_Cortex_M3.fvp_mps2.dma3**

ARM PrimeCell DMA Controller(PL080/081).
Type: `PL080_DMAC`.

**FVP_MPS2_Cortex_M3.fvp_mps2.dma3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_MPS2_Cortex_M3.fvp_mps2.dma3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_MPS2_Cortex_M3.fvp_mps2.dma3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_MPS2_Cortex_M3.fvp_mps2.dma3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M3.fvp_mps2.dma3_idau_labeller**

Type: `LabellerIdauSecurity`.

**FVP_MPS2_Cortex_M3.fvp_mps2.dma3_securitymodifier**

Type: `SecurityModifier`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M3.fvp_mps2.exclusive_monitor_psram**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_Cortex_M3.fvp_mps2.exclusive_monitor_psram_iotss**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_Cortex_M3.fvp_mps2.exclusive_monitor_switch_extra_psram_iotss**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M3.fvp_mps2.exclusive_monitor_switch_extra_psram_mps2**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M3.fvp_mps2.exclusive_monitor_zbtsram1**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_Cortex_M3.fvp_mps2.exclusive_monitor_zbtsram2**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_Cortex_M3.fvp_mps2.exclusive_squasher**

Squashes the exclusive attribute on bus transactions.
Type: `PVBusExclusiveSquasher`.

**FVP_MPS2_Cortex_M3.fvp_mps2.fpga_sysctrl**

FPGA SysCtrl timers LEDs and switches.
Type: `FPGA_SysCtrl`.

**FVP_MPS2_Cortex_M3.fvp_mps2.fpga_sysctrl.callBack100HzCounter**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M3.fvp_mps2.fpga_sysctrl.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M3.fvp_mps2.gpio_0_or_2**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_Cortex_M3.fvp_mps2.gpio_1_or_3**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_Cortex_M3.fvp_mps2.hostbridge**

Host Socket Interface Component.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `HostBridge`.

**FVP_MPS2_Cortex_M3.fvp_mps2.mem_switch_extra_psram_iotss**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M3.fvp_mps2.mem_switch_extra_psram_mps2**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M3.fvp_mps2.mpc_iotss_ssram1**

IoT Subsystem Memory Protection Controller.
Type: `IoTSS_MemoryProtectionController`.

**FVP_MPS2_Cortex_M3.fvp_mps2.mpc_iotss_ssram1.gating_disabled_thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M3.fvp_mps2.mpc_iotss_ssram2**

IoT Subsystem Memory Protection Controller.
Type: `IoTSS_MemoryProtectionController`.

**FVP_MPS2_Cortex_M3.fvp_mps2.mpc_iotss_ssram2.gating_disabled_thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M3.fvp_mps2.mpc_iotss_ssram3**

IoT Subsystem Memory Protection Controller.
Type: `IoTSS_MemoryProtectionController`.

**FVP_MPS2_Cortex_M3.fvp_mps2.mpc_iotss_ssram3.gating_disabled_thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M3.fvp_mps2.mps2_audio**

MPS2 Audio.
Type: `MPS2_Audio`.

**FVP_MPS2_Cortex_M3.fvp_mps2.mps2_cmsdk_dualtimer**

ARM Dual-Timer Module.
Type: `CMSDK_DualTimer`.

**FVP_MPS2_Cortex_M3.fvp_mps2.mps2_cmsdk_dualtimer.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M3.fvp_mps2.mps2_cmsdk_dualtimer.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M3.fvp_mps2.mps2_cmsdk_dualtimer.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M3.fvp_mps2.mps2_cmsdk_dualtimer.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M3.fvp_mps2.mps2_exclusive_monitor_zbtsram1**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_Cortex_M3.fvp_mps2.mps2_exclusive_monitor_zbtsram2**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_Cortex_M3.fvp_mps2.mps2_lcd**

MPS2 LCD I2C interface.
Type: `MPS2_LCD`.

**FVP_MPS2_Cortex_M3.fvp_mps2.mps2_mpc_iotss_ssram1**

IoT Subsystem Memory Protection Controller.
Type: `IoTSS_MemoryProtectionController`.

**FVP_MPS2_Cortex_M3.fvp_mps2.mps2_mpc_iotss_ssram1.gating_disabled_thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M3.fvp_mps2.mps2_secure_control_register_block**

MPS2 Secure Control Register Block.
Type: `MPS2_SecureCtrl`.

**FVP_MPS2_Cortex_M3.fvp_mps2.mps2_timer0**

ARM Timer Module.
Type: `CMSDK_Timer`.

**FVP_MPS2_Cortex_M3.fvp_mps2.mps2_timer0.clk_div**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M3.fvp_mps2.mps2_timer0.counter**

Internal component used by SP804 Timer module.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `CounterModule`.

**FVP_MPS2_Cortex_M3.fvp_mps2.mps2_timer1**

ARM Timer Module.
Type: `CMSDK_Timer`.

**FVP_MPS2_Cortex_M3.fvp_mps2.mps2_timer1.clk_div**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M3.fvp_mps2.mps2_timer1.counter**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M3.fvp_mps2.mps2_visualisation**

Display window for MPS2 using Visualisation library.
Type: `MPS2_Visualisation`.

**FVP_MPS2_Cortex_M3.fvp_mps2.niden_or_gate**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_Cortex_M3.fvp_mps2.nmi_or_gate**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_Cortex_M3.fvp_mps2.pl022_ssp_mps2**

ARM PrimeCell Synchronous Serial Port(PL022).
Type: `PL022_SSP_MPS2`.

**FVP_MPS2_Cortex_M3.fvp_mps2.pl022_ssp_mps2.prescaler**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M3.fvp_mps2.platform_bus_switch**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M3.fvp_mps2.platform_switch_dma0**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M3.fvp_mps2.platform_switch_dma1**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M3.fvp_mps2.signal_router**
Signal router.
Type: `SignalRouter`.

**FVP_MPS2_Cortex_M3.fvp_mps2.smsc_91c111**
SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_MPS2_Cortex_M3.fvp_mps2.spiden_or_gate**
Or Gate.
Type: `OrGate`.

**FVP_MPS2_Cortex_M3.fvp_mps2.spniden_or_gate**
Or Gate.
Type: `OrGate`.

**FVP_MPS2_Cortex_M3.fvp_mps2.sse200**
SSE-200 subsystem.
Type: `SSE200`.

**FVP_MPS2_Cortex_M3.fvp_mps2.sse200.acg_cpu0**
IoT Subsystem Access Control Gate.
Type: `IoTSS_AccessControlGate`.

**FVP_MPS2_Cortex_M3.fvp_mps2.sse200.acg_cpu1**
IoT Subsystem Access Control Gate.
Type: `IoTSS_AccessControlGate`.

**FVP_MPS2_Cortex_M3.fvp_mps2.sse200.acg_sram0**
IoT Subsystem Access Control Gate.
Type: `IoTSS_AccessControlGate`.

**FVP_MPS2_Cortex_M3.fvp_mps2.sse200.acg_sram1**
IoT Subsystem Access Control Gate.
Type: `IoTSS_AccessControlGate`.

**FVP_MPS2_Cortex_M3.fvp_mps2.sse200.acg_sram2**
IoT Subsystem Access Control Gate.
Type: `IoTSS_AccessControlGate`.

**FVP_MPS2_Cortex_M3.fvp_mps2.sse200.acg_sram3**
IoT Subsystem Access Control Gate.
Type: `IoTSS_AccessControlGate`.

**FVP_MPS2_Cortex_M3.fvp_mps2.sse200.apb_ppc_iotss_subsystem0**
IoT Subsystem Peripheral Protection Controller.
Type: `IoTSS_PeripheralProtectionController`.

**FVP_MPS2_Cortex_M3.fvp_mps2.sse200.apb_ppc_iotss_subsystem1**
IoT Subsystem Peripheral Protection Controller.
Type: `IoTSS_PeripheralProtectionController`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M3.fvp_mps2.sse200.clock32kHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M3.fvp_mps2.sse200.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M3.fvp_mps2.sse200.cmsdk_dualtimer**

ARM Dual-Timer Module.
Type: `CMSDK_DualTimer`.

**FVP_MPS2_Cortex_M3.fvp_mps2.sse200.cmsdk_dualtimer.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M3.fvp_mps2.sse200.cmsdk_dualtimer.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M3.fvp_mps2.sse200.cmsdk_dualtimer.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M3.fvp_mps2.sse200.cmsdk_dualtimer.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M3.fvp_mps2.sse200.cordio_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M3.fvp_mps2.sse200.cpu0core_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M3.fvp_mps2.sse200.cpu0dbg_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M3.fvp_mps2.sse200.cpu1core_ppu**
ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M3.fvp_mps2.sse200.cpu1dbg_ppu**
ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M3.fvp_mps2.sse200.crypto_ppu**
ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M3.fvp_mps2.sse200.dbg_ppu**
ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M3.fvp_mps2.sse200.exclusive_monitor_iotss_internal_sram0**
Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_Cortex_M3.fvp_mps2.sse200.exclusive_monitor_iotss_internal_sram1**
Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_Cortex_M3.fvp_mps2.sse200.exclusive_monitor_iotss_internal_sram2**
Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_Cortex_M3.fvp_mps2.sse200.exclusive_monitor_iotss_internal_sram3**
Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_Cortex_M3.fvp_mps2.sse200.exclusive_squasher**
Squashes the exclusive attribute on bus transactions.
Type: `PVBusExclusiveSquasher`.

**FVP_MPS2_Cortex_M3.fvp_mps2.sse200.idau_labeller**
Type: `LabellerIdauSecurity`.

**FVP_MPS2_Cortex_M3.fvp_mps2.sse200.internal_msniper_tcm**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M3.fvp_mps2.sse200.iotss_cpuidentity**
IoT Subsystem CPU_IDENTITY registers.
Type: `IoTSS_CPUIdentity`.

**FVP_MPS2_Cortex_M3.fvp_mps2.sse200.iotss_internal_sram0**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M3.fvp_mps2.sse200.iotss_internal_sram1**
RAM device, can be dynamic or static ram.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: RAMDevice.

**FVP_MPS2_Cortex_M3.fvp_mps2.sse200.iotss_internal_sram2**

RAM device, can be dynamic or static ram.
Type: RAMDevice.

**FVP_MPS2_Cortex_M3.fvp_mps2.sse200.iotss_internal_sram3**

RAM device, can be dynamic or static ram.
Type: RAMDevice.

**FVP_MPS2_Cortex_M3.fvp_mps2.sse200.iotss_systemcontrol**

IoT Subsystem System Control registers.
Type: IoTSS_SystemControl.

**FVP_MPS2_Cortex_M3.fvp_mps2.sse200.iotss_systeminfo**

IoT Subsystem System Information registers.
Type: IoTSS_SystemInfo.

**FVP_MPS2_Cortex_M3.fvp_mps2.sse200.mem_switch_internal_msniper_tcm**

Allow transactions to be routed arbitrarily.
Type: PVBusRouter.

**FVP_MPS2_Cortex_M3.fvp_mps2.sse200.mem_switch_internal_sram**

Allow transactions to be routed arbitrarily.
Type: PVBusRouter.

**FVP_MPS2_Cortex_M3.fvp_mps2.sse200.mem_switch_internal_sram_mpc**

Allow transactions to be routed arbitrarily.
Type: PVBusRouter.

**FVP_MPS2_Cortex_M3.fvp_mps2.sse200.mem_switch_ppu**

Allow transactions to be routed arbitrarily.
Type: PVBusRouter.

**FVP_MPS2_Cortex_M3.fvp_mps2.sse200.mhu0**

IoT Subsystem Message Handling Unit.
Type: IoTSS_MessageHandlingUnit.

**FVP_MPS2_Cortex_M3.fvp_mps2.sse200.mhu1**

IoT Subsystem Message Handling Unit.
Type: IoTSS_MessageHandlingUnit.

**FVP_MPS2_Cortex_M3.fvp_mps2.sse200.mpc_iotss_internal_sram0**

IoT Subsystem Memory Protection Controller.
Type: IoTSS_MemoryProtectionController.

**FVP_MPS2_Cortex_M3.fvp_mps2.sse200.mpc_iotss_internal_sram0.gating_disabled_thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: SchedulerThreadEvent.

**FVP_MPS2_Cortex_M3.fvp_mps2.sse200.mpc_iotss_internal_sram1**

IoT Subsystem Memory Protection Controller.
Type: IoTSS_MemoryProtectionController.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M3.fvp_mps2.sse200.mpc_iotss_internal_sram1.gating_disabled_thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M3.fvp_mps2.sse200.mpc_iotss_internal_sram2**

IoT Subsystem Memory Protection Controller.
Type: `IoTSS_MemoryProtectionController`.

**FVP_MPS2_Cortex_M3.fvp_mps2.sse200.mpc_iotss_internal_sram2.gating_disabled_thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M3.fvp_mps2.sse200.mpc_iotss_internal_sram3**

IoT Subsystem Memory Protection Controller.
Type: `IoTSS_MemoryProtectionController`.

**FVP_MPS2_Cortex_M3.fvp_mps2.sse200.mpc_iotss_internal_sram3.gating_disabled_thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M3.fvp_mps2.sse200.nmi_or_gate**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_Cortex_M3.fvp_mps2.sse200.nonsecure_watchdog**

ARM Watchdog Module.
Type: `CMSDK_Watchdog`.

**FVP_MPS2_Cortex_M3.fvp_mps2.sse200.ram0_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M3.fvp_mps2.sse200.ram1_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M3.fvp_mps2.sse200.ram2_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M3.fvp_mps2.sse200.ram3_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M3.fvp_mps2.sse200.s32k_timer**

ARM Timer Module.
Type: `CMSDK_Timer`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M3.fvp_mps2.sse200.s32k_timer.clk_div**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M3.fvp_mps2.sse200.s32k_timer.counter**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M3.fvp_mps2.sse200.s32k_watchdog**

ARM Watchdog Module.
Type: `CMSDK_Watchdog`.

**FVP_MPS2_Cortex_M3.fvp_mps2.sse200.secure_control_register_block**

MPS2 Secure Control Register Block.
Type: `MPS2_SecureCtrl`.

**FVP_MPS2_Cortex_M3.fvp_mps2.sse200.secure_watchdog**

ARM Watchdog Module.
Type: `CMSDK_Watchdog`.

**FVP_MPS2_Cortex_M3.fvp_mps2.sse200.signal_router**

Signal router.
Type: `SignalRouter`.

**FVP_MPS2_Cortex_M3.fvp_mps2.sse200.sys_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M3.fvp_mps2.sse200.timer0**

ARM Timer Module.
Type: `CMSDK_Timer`.

**FVP_MPS2_Cortex_M3.fvp_mps2.sse200.timer0.clk_div**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M3.fvp_mps2.sse200.timer0.counter**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M3.fvp_mps2.sse200.timer1**

ARM Timer Module.
Type: `CMSDK_Timer`.

**FVP_MPS2_Cortex_M3.fvp_mps2.sse200.timer1.clk_div**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M3.fvp_mps2.sse200.timer1.counter**
Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M3.fvp_mps2.ssram1**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M3.fvp_mps2.ssram2**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M3.fvp_mps2.stub0**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M3.fvp_mps2.stub1**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M3.fvp_mps2.stub_i2c1**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M3.fvp_mps2.stub_i2s**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M3.fvp_mps2.stub_spi0**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M3.fvp_mps2.stub_spi2**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M3.fvp_mps2.svos_dualtimer**
Type: `SVOS_DualTimer`.

**FVP_MPS2_Cortex_M3.fvp_mps2.svos_dualtimer.svos_dualtimer0**
ARM Dual-Timer Module.
Type: `CMSDK_DualTimer`.

**FVP_MPS2_Cortex_M3.fvp_mps2.svos_dualtimer.svos_dualtimer0.clk_div0**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M3.fvp_mps2.svos_dualtimer.svos_dualtimer0.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M3.fvp_mps2.svos_dualtimer.svos_dualtimer0.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M3.fvp_mps2.svos_dualtimer.svos_dualtimer0.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M3.fvp_mps2.svos_dualtimer.svos_dualtimer1**

ARM Dual-Timer Module.
Type: `CMSDK_DualTimer`.

**FVP_MPS2_Cortex_M3.fvp_mps2.svos_dualtimer.svos_dualtimer1.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M3.fvp_mps2.svos_dualtimer.svos_dualtimer1.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M3.fvp_mps2.svos_dualtimer.svos_dualtimer1.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M3.fvp_mps2.svos_dualtimer.svos_dualtimer1.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M3.fvp_mps2.svos_dualtimer.svos_dualtimer2**

ARM Dual-Timer Module.
Type: `CMSDK_DualTimer`.

**FVP_MPS2_Cortex_M3.fvp_mps2.svos_dualtimer.svos_dualtimer2.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M3.fvp_mps2.svos_dualtimer.svos_dualtimer2.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M3.fvp_mps2.svos_dualtimer.svos_dualtimer2.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M3.fvp_mps2.svos_dualtimer.svos_dualtimer2.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M3.fvp_mps2.svos_dualtimer.svos_dualtimer3**

ARM Dual-Timer Module.
Type: `CMSDK_DualTimer`.

**FVP_MPS2_Cortex_M3.fvp_mps2.svos_dualtimer.svos_dualtimer3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M3.fvp_mps2.svos_dualtimer.svos_dualtimer3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M3.fvp_mps2.svos_dualtimer.svos_dualtimer3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M3.fvp_mps2.svos_dualtimer.svos_dualtimer3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M3.fvp_mps2.switch_PSRAM_M7**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M3.fvp_mps2.switch_svos_dualtimer**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M3.fvp_mps2.telnetterminal0**

Telnet terminal interface.
Type: `TelnetTerminal`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M3.fvp_mps2.telnetterminal1**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_MPS2_Cortex_M3.fvp_mps2.telnetterminal2**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_MPS2_Cortex_M3.fvp_mps2.touchscreen_interface**
MPS2 Touch Screen.
Type: `MPS2_TouchScreen`.

**FVP_MPS2_Cortex_M3.fvp_mps2.uart_overflows_or_gate**
Or Gate.
Type: `OrGate`.

## 16.6 FVP_MPS2_Cortex-M33

List of instances in FVP_MPS2_Cortex-M33.

### FVP_MPS2_Cortex-M33 instances

**FVP_MPS2_Cortex_M33**
Type: `FVP_MPS2_Cortex_M33`.

**FVP_MPS2_Cortex_M33.clk25Mhz**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M33.clk25khz**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M33.cpu0**
ARM CORTEXM33 CT model.
Type: `ARM_Cortex-M33`.

**FVP_MPS2_Cortex_M33.cpu1**
ARM CORTEXM33 CT model.
Type: `ARM_Cortex-M33`.

**FVP_MPS2_Cortex_M33.fvp_mps2**
MPS2 DUT.
Type: `FVP_MPS2`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M33.fvp_mps2.GPIO0**

ARM PrimeCell General Purpose Input/Output.
Type: `CMSDK_GPIO`.

**FVP_MPS2_Cortex_M33.fvp_mps2.GPIO1**

ARM PrimeCell General Purpose Input/Output.
Type: `CMSDK_GPIO`.

**FVP_MPS2_Cortex_M33.fvp_mps2.GPIO2**

ARM PrimeCell General Purpose Input/Output.
Type: `CMSDK_GPIO`.

**FVP_MPS2_Cortex_M33.fvp_mps2.GPIO3**

ARM PrimeCell General Purpose Input/Output.
Type: `CMSDK_GPIO`.

**FVP_MPS2_Cortex_M33.fvp_mps2.GPIO_connection_test**

Type: `GPIO_Connection_Test`.

**FVP_MPS2_Cortex_M33.fvp_mps2.GPIO_connection_test.GPIO0_port_trans**

Type: `GPIO_Port_Transfer`.

**FVP_MPS2_Cortex_M33.fvp_mps2.GPIO_connection_test.GPIO1_port_test**

Type: `GPIO1_Connection_Test`.

**FVP_MPS2_Cortex_M33.fvp_mps2.PSRAM**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M33.fvp_mps2.PSRAM_M7**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M33.fvp_mps2.UART0**

ARM CMSDK UART Module.
Type: `CMSDK_UART`.

**FVP_MPS2_Cortex_M33.fvp_mps2.UART0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M33.fvp_mps2.UART1**

ARM CMSDK UART Module.
Type: `CMSDK_UART`.

**FVP_MPS2_Cortex_M33.fvp_mps2.UART1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ClockDivider`.

**FVP_MPS2_Cortex_M33.fvp_mps2.UART2**

ARM CMSDK UART Module.
Type: `CMSDK_UART`.

**FVP_MPS2_Cortex_M33.fvp_mps2.UART2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M33.fvp_mps2.VGA_interface**

VGA display interface between main bus and visualisation.
Type: `MPS2_VGA`.

**FVP_MPS2_Cortex_M33.fvp_mps2.ahb_ppc_iotss_expansion0**

IoT Subsystem Peripheral Protection Controller.
Type: `IoTSS_PeripheralProtectionController`.

**FVP_MPS2_Cortex_M33.fvp_mps2.ahb_ppc_iotss_expansion1**

IoT Subsystem Peripheral Protection Controller.
Type: `IoTSS_PeripheralProtectionController`.

**FVP_MPS2_Cortex_M33.fvp_mps2.apb_ppc_iotss_expansion0**

IoT Subsystem Peripheral Protection Controller.
Type: `IoTSS_PeripheralProtectionController`.

**FVP_MPS2_Cortex_M33.fvp_mps2.apb_ppc_iotss_expansion1**

IoT Subsystem Peripheral Protection Controller.
Type: `IoTSS_PeripheralProtectionController`.

**FVP_MPS2_Cortex_M33.fvp_mps2.apb_ppc_iotss_expansion2**

IoT Subsystem Peripheral Protection Controller.
Type: `IoTSS_PeripheralProtectionController`.

**FVP_MPS2_Cortex_M33.fvp_mps2.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M33.fvp_mps2.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M33.fvp_mps2.cmsdk_sysctrl**

Cortex-M Simple System Control.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `CMSDK_SysCtrl`.

**FVP_MPS2_Cortex_M33.fvp_mps2.cmsdk_watchdog**

ARM Watchdog Module.
Type: `CMSDK_Watchdog`.

**FVP_MPS2_Cortex_M33.fvp_mps2.cpu_wait_or_gate_0**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_Cortex_M33.fvp_mps2.cpu_wait_or_gate_1**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_Cortex_M33.fvp_mps2.dbgen_or_gate**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_Cortex_M33.fvp_mps2.dma0**

ARM PrimeCell DMA Controller(PL080/081).
Type: `PL080_DMAC`.

**FVP_MPS2_Cortex_M33.fvp_mps2.dma0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_MPS2_Cortex_M33.fvp_mps2.dma0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_MPS2_Cortex_M33.fvp_mps2.dma0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_MPS2_Cortex_M33.fvp_mps2.dma0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M33.fvp_mps2.dma0_idau_labeller**

Type: `LabellerIdauSecurity`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M33.fvp_mps2.dma0_securitymodifier**

Type: `SecurityModifier`.

**FVP_MPS2_Cortex_M33.fvp_mps2.dma1**

ARM PrimeCell DMA Controller(PL080/081).
Type: `PL080_DMAC`.

**FVP_MPS2_Cortex_M33.fvp_mps2.dma1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_MPS2_Cortex_M33.fvp_mps2.dma1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_MPS2_Cortex_M33.fvp_mps2.dma1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_MPS2_Cortex_M33.fvp_mps2.dma1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M33.fvp_mps2.dma1_idau_labeller**

Type: `LabellerIdauSecurity`.

**FVP_MPS2_Cortex_M33.fvp_mps2.dma1_securitymodifier**

Type: `SecurityModifier`.

**FVP_MPS2_Cortex_M33.fvp_mps2.dma2**

ARM PrimeCell DMA Controller(PL080/081).
Type: `PL080_DMAC`.

**FVP_MPS2_Cortex_M33.fvp_mps2.dma2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_MPS2_Cortex_M33.fvp_mps2.dma2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_MPS2_Cortex_M33.fvp_mps2.dma2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_MPS2_Cortex_M33.fvp_mps2.dma2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M33.fvp_mps2.dma2_idau_labeller**

Type: `LabellerIdauSecurity`.

**FVP_MPS2_Cortex_M33.fvp_mps2.dma2_securitymodifier**

Type: `SecurityModifier`.

**FVP_MPS2_Cortex_M33.fvp_mps2.dma3**

ARM PrimeCell DMA Controller(PL080/081).
Type: `PL080_DMAC`.

**FVP_MPS2_Cortex_M33.fvp_mps2.dma3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_MPS2_Cortex_M33.fvp_mps2.dma3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M33.fvp_mps2.dma3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_MPS2_Cortex_M33.fvp_mps2.dma3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M33.fvp_mps2.dma3_idau_labeller**

Type: `LabellerIdauSecurity`.

**FVP_MPS2_Cortex_M33.fvp_mps2.dma3_securitymodifier**

Type: `SecurityModifier`.

**FVP_MPS2_Cortex_M33.fvp_mps2.exclusive_monitor_psram**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_Cortex_M33.fvp_mps2.exclusive_monitor_psram_iotss**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_Cortex_M33.fvp_mps2.exclusive_monitor_switch_extra_psram_iotss**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M33.fvp_mps2.exclusive_monitor_switch_extra_psram_mps2**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M33.fvp_mps2.exclusive_monitor_zbtsram1**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_Cortex_M33.fvp_mps2.exclusive_monitor_zbtsram2**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_Cortex_M33.fvp_mps2.exclusive_squasher**

Squashes the exclusive attribute on bus transactions.
Type: `PVBusExclusiveSquasher`.

**FVP_MPS2_Cortex_M33.fvp_mps2.fpga_sysctrl**

FPGA SysCtrl timers LEDs and switches.
Type: `FPGA_SysCtrl`.

**FVP_MPS2_Cortex_M33.fvp_mps2.fpga_sysctrl.callBack100HzCounter**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M33.fvp_mps2.fpga_sysctrl.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M33.fvp_mps2.gpio_0_or_2**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_Cortex_M33.fvp_mps2.gpio_1_or_3**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_Cortex_M33.fvp_mps2.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_MPS2_Cortex_M33.fvp_mps2.mem_switch_extra_psram_iotss**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M33.fvp_mps2.mem_switch_extra_psram_mps2**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M33.fvp_mps2.mpc_iotss_ssram1**

IoT Subsystem Memory Protection Controller.
Type: `IoTSS_MemoryProtectionController`.

**FVP_MPS2_Cortex_M33.fvp_mps2.mpc_iotss_ssram1.gating_disabled_thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M33.fvp_mps2.mpc_iotss_ssram2**

IoT Subsystem Memory Protection Controller.
Type: `IoTSS_MemoryProtectionController`.

**FVP_MPS2_Cortex_M33.fvp_mps2.mpc_iotss_ssram2.gating_disabled_thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M33.fvp_mps2.mpc_iotss_ssram3**

IoT Subsystem Memory Protection Controller.
Type: `IoTSS_MemoryProtectionController`.

**FVP_MPS2_Cortex_M33.fvp_mps2.mpc_iotss_ssram3.gating_disabled_thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M33.fvp_mps2.mps2_audio**

MPS2 Audio.
Type: `MPS2_Audio`.

**FVP_MPS2_Cortex_M33.fvp_mps2.mps2_cmsdk_dualtimer**

ARM Dual-Timer Module.
Type: `CMSDK_DualTimer`.

**FVP_MPS2_Cortex_M33.fvp_mps2.mps2_cmsdk_dualtimer.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M33.fvp_mps2.mps2_cmsdk_dualtimer.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M33.fvp_mps2.mps2_cmsdk_dualtimer.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M33.fvp_mps2.mps2_cmsdk_dualtimer.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M33.fvp_mps2.mps2_exclusive_monitor_zbtsram1**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_Cortex_M33.fvp_mps2.mps2_exclusive_monitor_zbtsram2**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_Cortex_M33.fvp_mps2.mps2_lcd**

MPS2 LCD I2C interface.
Type: `MPS2_LCD`.

**FVP_MPS2_Cortex_M33.fvp_mps2.mps2_mpc_iotss_ssram1**

IoT Subsystem Memory Protection Controller.
Type: `IoTSS_MemoryProtectionController`.

**FVP_MPS2_Cortex_M33.fvp_mps2.mps2_mpc_iotss_ssram1.gating_disabled_thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M33.fvp_mps2.mps2_secure_control_register_block**

MPS2 Secure Control Register Block.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `MPS2_SecureCtrl`.

**FVP_MPS2_Cortex_M33.fvp_mps2.mps2_timer0**

ARM Timer Module.
Type: `CMSDK_Timer`.

**FVP_MPS2_Cortex_M33.fvp_mps2.mps2_timer0.clk_div**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M33.fvp_mps2.mps2_timer0.counter**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M33.fvp_mps2.mps2_timer1**

ARM Timer Module.
Type: `CMSDK_Timer`.

**FVP_MPS2_Cortex_M33.fvp_mps2.mps2_timer1.clk_div**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M33.fvp_mps2.mps2_timer1.counter**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M33.fvp_mps2.mps2_visualisation**

Display window for MPS2 using Visualisation library.
Type: `MPS2_Visualisation`.

**FVP_MPS2_Cortex_M33.fvp_mps2.niden_or_gate**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_Cortex_M33.fvp_mps2.nmi_or_gate**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_Cortex_M33.fvp_mps2.pl022_ssp_mps2**

ARM PrimeCell Synchronous Serial Port(PL022).
Type: `PL022_SSP_MPS2`.

**FVP_MPS2_Cortex_M33.fvp_mps2.pl022_ssp_mps2.prescaler**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ClockDivider`.

**FVP_MPS2_Cortex_M33.fvp_mps2.platform_bus_switch**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M33.fvp_mps2.platform_switch_dma0**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M33.fvp_mps2.platform_switch_dma1**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M33.fvp_mps2.signal_router**

Signal router.
Type: `SignalRouter`.

**FVP_MPS2_Cortex_M33.fvp_mps2.smsc_91c111**

SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_MPS2_Cortex_M33.fvp_mps2.spiden_or_gate**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_Cortex_M33.fvp_mps2.spniden_or_gate**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_Cortex_M33.fvp_mps2.sse200**

SSE-200 subsystem.
Type: `SSE200`.

**FVP_MPS2_Cortex_M33.fvp_mps2.sse200.acg_cpu0**

IoT Subsystem Access Control Gate.
Type: `IoTSS_AccessControlGate`.

**FVP_MPS2_Cortex_M33.fvp_mps2.sse200.acg_cpu1**

IoT Subsystem Access Control Gate.
Type: `IoTSS_AccessControlGate`.

**FVP_MPS2_Cortex_M33.fvp_mps2.sse200.acg_sram0**

IoT Subsystem Access Control Gate.
Type: `IoTSS_AccessControlGate`.

**FVP_MPS2_Cortex_M33.fvp_mps2.sse200.acg_sram1**

IoT Subsystem Access Control Gate.
Type: `IoTSS_AccessControlGate`.

**FVP_MPS2_Cortex_M33.fvp_mps2.sse200.acg_sram2**

IoT Subsystem Access Control Gate.
Type: `IoTSS_AccessControlGate`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M33.fvp_mps2.sse200.acg_sram3**

IoT Subsystem Access Control Gate.
Type: `IoTSS_AccessControlGate`.

**FVP_MPS2_Cortex_M33.fvp_mps2.sse200.apb_ppc_iotss_subsystem0**

IoT Subsystem Peripheral Protection Controller.
Type: `IoTSS_PeripheralProtectionController`.

**FVP_MPS2_Cortex_M33.fvp_mps2.sse200.apb_ppc_iotss_subsystem1**

IoT Subsystem Peripheral Protection Controller.
Type: `IoTSS_PeripheralProtectionController`.

**FVP_MPS2_Cortex_M33.fvp_mps2.sse200.clock32kHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M33.fvp_mps2.sse200.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M33.fvp_mps2.sse200.cmsdk_dualtimer**

ARM Dual-Timer Module.
Type: `CMSDK_DualTimer`.

**FVP_MPS2_Cortex_M33.fvp_mps2.sse200.cmsdk_dualtimer.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M33.fvp_mps2.sse200.cmsdk_dualtimer.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M33.fvp_mps2.sse200.cmsdk_dualtimer.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M33.fvp_mps2.sse200.cmsdk_dualtimer.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M33.fvp_mps2.sse200.cordio_ppu**
ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M33.fvp_mps2.sse200.cpu0core_ppu**
ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M33.fvp_mps2.sse200.cpu0dbg_ppu**
ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M33.fvp_mps2.sse200.cpu1core_ppu**
ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M33.fvp_mps2.sse200.cpu1dbg_ppu**
ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M33.fvp_mps2.sse200.crypto_ppu**
ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M33.fvp_mps2.sse200.dbg_ppu**
ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M33.fvp_mps2.sse200.exclusive_monitor_iotss_internal_sram0**
Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_Cortex_M33.fvp_mps2.sse200.exclusive_monitor_iotss_internal_sram1**
Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_Cortex_M33.fvp_mps2.sse200.exclusive_monitor_iotss_internal_sram2**
Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_Cortex_M33.fvp_mps2.sse200.exclusive_monitor_iotss_internal_sram3**
Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_Cortex_M33.fvp_mps2.sse200.exclusive_squasher**
Squashes the exclusive attribute on bus transactions.
Type: `PVBusExclusiveSquasher`.

**FVP_MPS2_Cortex_M33.fvp_mps2.sse200.idau_labeller**
Type: `LabellerIdauSecurity`.

**FVP_MPS2_Cortex_M33.fvp_mps2.sse200.internal_msniper_tcm**
RAM device, can be dynamic or static ram.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `RAMDevice`.

**FVP_MPS2_Cortex_M33.fvp_mps2.sse200.iotss_cpuidentity**

IoT Subsystem CPU_IDENTITY registers.
Type: `IoTSS_CPUIdentity`.

**FVP_MPS2_Cortex_M33.fvp_mps2.sse200.iotss_internal_sram0**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M33.fvp_mps2.sse200.iotss_internal_sram1**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M33.fvp_mps2.sse200.iotss_internal_sram2**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M33.fvp_mps2.sse200.iotss_internal_sram3**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M33.fvp_mps2.sse200.iotss_systemcontrol**

IoT Subsystem System Control registers.
Type: `IoTSS_SystemControl`.

**FVP_MPS2_Cortex_M33.fvp_mps2.sse200.iotss_systeminfo**

IoT Subsystem System Information registers.
Type: `IoTSS_SystemInfo`.

**FVP_MPS2_Cortex_M33.fvp_mps2.sse200.mem_switch_internal_msniper_tcm**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M33.fvp_mps2.sse200.mem_switch_internal_sram**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M33.fvp_mps2.sse200.mem_switch_internal_sram_mpc**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M33.fvp_mps2.sse200.mem_switch_ppu**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M33.fvp_mps2.sse200.mhu0**

IoT Subsystem Message Handling Unit.
Type: `IoTSS_MessageHandlingUnit`.

**FVP_MPS2_Cortex_M33.fvp_mps2.sse200.mhu1**

IoT Subsystem Message Handling Unit.
Type: `IoTSS_MessageHandlingUnit`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M33.fvp_mps2.sse200.mpc_iotss_internal_sram0**

IoT Subsystem Memory Protection Controller.
Type: `IoTSS_MemoryProtectionController`.

**FVP_MPS2_Cortex_M33.fvp_mps2.sse200.mpc_iotss_internal_sram0.gating_disabled_thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M33.fvp_mps2.sse200.mpc_iotss_internal_sram1**

IoT Subsystem Memory Protection Controller.
Type: `IoTSS_MemoryProtectionController`.

**FVP_MPS2_Cortex_M33.fvp_mps2.sse200.mpc_iotss_internal_sram1.gating_disabled_thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M33.fvp_mps2.sse200.mpc_iotss_internal_sram2**

IoT Subsystem Memory Protection Controller.
Type: `IoTSS_MemoryProtectionController`.

**FVP_MPS2_Cortex_M33.fvp_mps2.sse200.mpc_iotss_internal_sram2.gating_disabled_thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M33.fvp_mps2.sse200.mpc_iotss_internal_sram3**

IoT Subsystem Memory Protection Controller.
Type: `IoTSS_MemoryProtectionController`.

**FVP_MPS2_Cortex_M33.fvp_mps2.sse200.mpc_iotss_internal_sram3.gating_disabled_thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M33.fvp_mps2.sse200.nmi_or_gate**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_Cortex_M33.fvp_mps2.sse200.nonsecure_watchdog**

ARM Watchdog Module.
Type: `CMSDK_Watchdog`.

**FVP_MPS2_Cortex_M33.fvp_mps2.sse200.ram0_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M33.fvp_mps2.sse200.ram1_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M33.fvp_mps2.sse200.ram2_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M33.fvp_mps2.sse200.ram3_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M33.fvp_mps2.sse200.s32k_timer**

ARM Timer Module.
Type: `CMSDK_Timer`.

**FVP_MPS2_Cortex_M33.fvp_mps2.sse200.s32k_timer.clk_div**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M33.fvp_mps2.sse200.s32k_timer.counter**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M33.fvp_mps2.sse200.s32k_watchdog**

ARM Watchdog Module.
Type: `CMSDK_Watchdog`.

**FVP_MPS2_Cortex_M33.fvp_mps2.sse200.secure_control_register_block**

MPS2 Secure Control Register Block.
Type: `MPS2_SecureCtrl`.

**FVP_MPS2_Cortex_M33.fvp_mps2.sse200.secure_watchdog**

ARM Watchdog Module.
Type: `CMSDK_Watchdog`.

**FVP_MPS2_Cortex_M33.fvp_mps2.sse200.signal_router**

Signal router.
Type: `SignalRouter`.

**FVP_MPS2_Cortex_M33.fvp_mps2.sse200.sys_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M33.fvp_mps2.sse200.timer0**

ARM Timer Module.
Type: `CMSDK_Timer`.

**FVP_MPS2_Cortex_M33.fvp_mps2.sse200.timer0.clk_div**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M33.fvp_mps2.sse200.timer0.counter**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M33.fvp_mps2.sse200.timer1**

ARM Timer Module.
Type: `CMSDK_Timer`.

**FVP_MPS2_Cortex_M33.fvp_mps2.sse200.timer1.clk_div**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M33.fvp_mps2.sse200.timer1.counter**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M33.fvp_mps2.ssram1**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M33.fvp_mps2.ssram2**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M33.fvp_mps2.stub0**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M33.fvp_mps2.stub1**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M33.fvp_mps2.stub_i2c1**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M33.fvp_mps2.stub_i2s**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M33.fvp_mps2.stub_spi0**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M33.fvp_mps2.stub_spi2**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M33.fvp_mps2.svos_dualtimer**

Type: `SVOS_DualTimer`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M33.fvp_mps2.svos_dualtimer.svos_dualtimer0**

ARM Dual-Timer Module.
Type: `CMSDK_DualTimer`.

**FVP_MPS2_Cortex_M33.fvp_mps2.svos_dualtimer.svos_dualtimer0.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M33.fvp_mps2.svos_dualtimer.svos_dualtimer0.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M33.fvp_mps2.svos_dualtimer.svos_dualtimer0.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M33.fvp_mps2.svos_dualtimer.svos_dualtimer0.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M33.fvp_mps2.svos_dualtimer.svos_dualtimer1**

ARM Dual-Timer Module.
Type: `CMSDK_DualTimer`.

**FVP_MPS2_Cortex_M33.fvp_mps2.svos_dualtimer.svos_dualtimer1.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M33.fvp_mps2.svos_dualtimer.svos_dualtimer1.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M33.fvp_mps2.svos_dualtimer.svos_dualtimer1.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M33.fvp_mps2.svos_dualtimer.svos_dualtimer1.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M33.fvp_mps2.svos_dualtimer.svos_dualtimer2**

ARM Dual-Timer Module.
Type: `CMSDK_DualTimer`.

**FVP_MPS2_Cortex_M33.fvp_mps2.svos_dualtimer.svos_dualtimer2.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M33.fvp_mps2.svos_dualtimer.svos_dualtimer2.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M33.fvp_mps2.svos_dualtimer.svos_dualtimer2.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M33.fvp_mps2.svos_dualtimer.svos_dualtimer2.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M33.fvp_mps2.svos_dualtimer.svos_dualtimer3**

ARM Dual-Timer Module.
Type: `CMSDK_DualTimer`.

**FVP_MPS2_Cortex_M33.fvp_mps2.svos_dualtimer.svos_dualtimer3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M33.fvp_mps2.svos_dualtimer.svos_dualtimer3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M33.fvp_mps2.svos_dualtimer.svos_dualtimer3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M33.fvp_mps2.svos_dualtimer.svos_dualtimer3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M33.fvp_mps2.switch_PSRAM_M7**
Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M33.fvp_mps2.switch_svos_dualtimer**
Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M33.fvp_mps2.telnetterminal0**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_MPS2_Cortex_M33.fvp_mps2.telnetterminal1**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_MPS2_Cortex_M33.fvp_mps2.telnetterminal2**
Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_MPS2_Cortex_M33.fvp_mps2.touchscreen_interface**
MPS2 Touch Screen.
Type: `MPS2_TouchScreen`.

**FVP_MPS2_Cortex_M33.fvp_mps2.uart_overflows_or_gate**
Or Gate.
Type: `OrGate`.

## 16.7 FVP_MPS2_Cortex-M35P

List of instances in FVP_MPS2_Cortex-M35P.

### FVP_MPS2_Cortex-M35P instances

**FVP_MPS2_Cortex_M35P**
Type: `FVP_MPS2_Cortex_M35P`.

**FVP_MPS2_Cortex_M35P.clk25Mhz**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M35P.clk25khz**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M35P.cpu0**

ARM CORTEXM35P CT model.
Type: ARM_Cortex-M35P.

**FVP_MPS2_Cortex_M35P.cpu1**

ARM CORTEXM35P CT model.
Type: ARM_Cortex-M35P.

**FVP_MPS2_Cortex_M35P.fvp_mps2**

MPS2 DUT.
Type: FVP_MPS2.

**FVP_MPS2_Cortex_M35P.fvp_mps2.GPIO0**

ARM PrimeCell General Purpose Input/Output.
Type: CMSDK_GPIO.

**FVP_MPS2_Cortex_M35P.fvp_mps2.GPIO1**

ARM PrimeCell General Purpose Input/Output.
Type: CMSDK_GPIO.

**FVP_MPS2_Cortex_M35P.fvp_mps2.GPIO2**

ARM PrimeCell General Purpose Input/Output.
Type: CMSDK_GPIO.

**FVP_MPS2_Cortex_M35P.fvp_mps2.GPIO3**

ARM PrimeCell General Purpose Input/Output.
Type: CMSDK_GPIO.

**FVP_MPS2_Cortex_M35P.fvp_mps2.GPIO_connection_test**

Type: GPIO_Connection_Test.

**FVP_MPS2_Cortex_M35P.fvp_mps2.GPIO_connection_test.GPIO0_port_trans**

Type: GPIO_Port_Transfer.

**FVP_MPS2_Cortex_M35P.fvp_mps2.GPIO_connection_test.GPIO1_port_test**

Type: GPIO1_Connection_Test.

**FVP_MPS2_Cortex_M35P.fvp_mps2.PSRAM**

RAM device, can be dynamic or static ram.
Type: RAMDevice.

**FVP_MPS2_Cortex_M35P.fvp_mps2.PSRAM_M7**

RAM device, can be dynamic or static ram.
Type: RAMDevice.

**FVP_MPS2_Cortex_M35P.fvp_mps2.UART0**

ARM CMSDK UART Module.
Type: CMSDK_UART.

**FVP_MPS2_Cortex_M35P.fvp_mps2.UART0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.UART1**

ARM CMSDK UART Module.
Type: `CMSDK_UART`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.UART1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.UART2**

ARM CMSDK UART Module.
Type: `CMSDK_UART`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.UART2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.VGA_interface**

VGA display interface between main bus and visualisation.
Type: `MPS2_VGA`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.ahb_ppc_iotss_expansion0**

IoT Subsystem Peripheral Protection Controller.
Type: `IoTSS_PeripheralProtectionController`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.ahb_ppc_iotss_expansion1**

IoT Subsystem Peripheral Protection Controller.
Type: `IoTSS_PeripheralProtectionController`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.apb_ppc_iotss_expansion0**

IoT Subsystem Peripheral Protection Controller.
Type: `IoTSS_PeripheralProtectionController`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.apb_ppc_iotss_expansion1**

IoT Subsystem Peripheral Protection Controller.
Type: `IoTSS_PeripheralProtectionController`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.apb_ppc_iotss_expansion2**

IoT Subsystem Peripheral Protection Controller.
Type: `IoTSS_PeripheralProtectionController`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.cmsdk_sysctrl**

Cortex-M Simple System Control.
Type: `CMSDK_SysCtrl`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.cmsdk_watchdog**

ARM Watchdog Module.
Type: `CMSDK_Watchdog`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.cpu_wait_or_gate_0**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.cpu_wait_or_gate_1**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.dbgen_or_gate**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.dma0**

ARM PrimeCell DMA Controller(PL080/081).
Type: `PL080_DMAC`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.dma0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.dma0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ClockTimerThread64`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.dma0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.dma0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.dma0_idau_labeller**

Type: `LabellerIdauSecurity`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.dma0_securitymodifier**

Type: `SecurityModifier`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.dma1**

ARM PrimeCell DMA Controller(PL080/081).
Type: `PL080_DMAC`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.dma1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.dma1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.dma1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.dma1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.dma1_idau_labeller**

Type: `LabellerIdauSecurity`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M35P.fvp_mps2.dma1_securitymodifier**

Type: `SecurityModifier`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.dma2**

ARM PrimeCell DMA Controller(PL080/081).
Type: `PL080_DMAC`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.dma2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.dma2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.dma2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.dma2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.dma2_idau_labeller**

Type: `LabellerIdauSecurity`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.dma2_securitymodifier**

Type: `SecurityModifier`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.dma3**

ARM PrimeCell DMA Controller(PL080/081).
Type: `PL080_DMAC`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.dma3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.dma3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.dma3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.dma3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.dma3_idau_labeller**

Type: `LabellerIdauSecurity`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.dma3_securitymodifier**

Type: `SecurityModifier`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.exclusive_monitor_psram**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.exclusive_monitor_psram_iotss**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.exclusive_monitor_switch_extra_psram_iotss**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.exclusive_monitor_switch_extra_psram_mps2**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.exclusive_monitor_zbtsram1**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.exclusive_monitor_zbtsram2**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M35P.fvp_mps2.exclusive_squasher**

Squashes the exclusive attribute on bus transactions.
Type: `PVBusExclusiveSquasher`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.fpga_sysctrl**

FPGA SysCtrl timers LEDs and switches.
Type: `FPGA_SysCtrl`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.fpga_sysctrl.callBack100HzCounter**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.fpga_sysctrl.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.gpio_0_or_2**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.gpio_1_or_3**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.mem_switch_extra_psram_iotss**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.mem_switch_extra_psram_mps2**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.mpc_iotss_ssram1**

IoT Subsystem Memory Protection Controller.
Type: `IoTSS_MemoryProtectionController`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.mpc_iotss_ssram1.gating_disabled_thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.mpc_iotss_ssram2**

IoT Subsystem Memory Protection Controller.
Type: `IoTSS_MemoryProtectionController`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M35P.fvp_mps2.mpc_iotss_ssram2.gating_disabled_thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.mpc_iotss_ssram3**

IoT Subsystem Memory Protection Controller.
Type: `IoTSS_MemoryProtectionController`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.mpc_iotss_ssram3.gating_disabled_thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.mps2_audio**

MPS2 Audio.
Type: `MPS2_Audio`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.mps2_cmsdk_dualtimer**

ARM Dual-Timer Module.
Type: `CMSDK_DualTimer`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.mps2_cmsdk_dualtimer.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.mps2_cmsdk_dualtimer.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.mps2_cmsdk_dualtimer.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.mps2_cmsdk_dualtimer.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.mps2_exclusive_monitor_zbtsram1**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.mps2_exclusive_monitor_zbtsram2**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M35P.fvp_mps2.mps2_lcd**

MPS2 LCD I2C interface.
Type: `MPS2_LCD`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.mps2_mpc_iotss_ssram1**

IoT Subsystem Memory Protection Controller.
Type: `IoTSS_MemoryProtectionController`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.mps2_mpc_iotss_ssram1.gating_disabled_thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.mps2_secure_control_register_block**

MPS2 Secure Control Register Block.
Type: `MPS2_SecureCtrl`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.mps2_timer0**

ARM Timer Module.
Type: `CMSDK_Timer`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.mps2_timer0.clk_div**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.mps2_timer0.counter**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.mps2_timer1**

ARM Timer Module.
Type: `CMSDK_Timer`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.mps2_timer1.clk_div**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.mps2_timer1.counter**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.mps2_visualisation**

Display window for MPS2 using Visualisation library.
Type: `MPS2_Visualisation`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.niden_or_gate**

Or Gate.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `OrGate`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.nmi_or_gate**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.pl022_ssp_mps2**

ARM PrimeCell Synchronous Serial Port(PL022).
Type: `PL022_SSP_MPS2`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.pl022_ssp_mps2.prescaler**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.platform_bus_switch**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.platform_switch_dma0**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.platform_switch_dma1**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.signal_router**

Signal router.
Type: `SignalRouter`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.smsc_91c111**

SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.spiden_or_gate**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.spniden_or_gate**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.sse200**

SSE-200 subsystem.
Type: `SSE200`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.sse200.acg_cpu0**

IoT Subsystem Access Control Gate.
Type: `IoTSS_AccessControlGate`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M35P.fvp_mps2.sse200.acg_cpu1**
IoT Subsystem Access Control Gate.
Type: `IoTSS_AccessControlGate`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.sse200.acg_sram0**
IoT Subsystem Access Control Gate.
Type: `IoTSS_AccessControlGate`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.sse200.acg_sram1**
IoT Subsystem Access Control Gate.
Type: `IoTSS_AccessControlGate`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.sse200.acg_sram2**
IoT Subsystem Access Control Gate.
Type: `IoTSS_AccessControlGate`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.sse200.acg_sram3**
IoT Subsystem Access Control Gate.
Type: `IoTSS_AccessControlGate`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.sse200.apb_ppc_iotss_subsystem0**
IoT Subsystem Peripheral Protection Controller.
Type: `IoTSS_PeripheralProtectionController`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.sse200.apb_ppc_iotss_subsystem1**
IoT Subsystem Peripheral Protection Controller.
Type: `IoTSS_PeripheralProtectionController`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.sse200.clock32kHz**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.sse200.clockdivider**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.sse200.cmsdk_dualtimer**
ARM Dual-Timer Module.
Type: `CMSDK_DualTimer`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.sse200.cmsdk_dualtimer.clk_div0**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M35P.fvp_mps2.sse200.cmsdk_dualtimer.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.sse200.cmsdk_dualtimer.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.sse200.cmsdk_dualtimer.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.sse200.cordio_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.sse200.cpu0core_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.sse200.cpu0dbg_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.sse200.cpu1core_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.sse200.cpu1dbg_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.sse200.crypto_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.sse200.dbg_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.sse200.exclusive_monitor_iotss_internal_sram0**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.sse200.exclusive_monitor_iotss_internal_sram1**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.sse200.exclusive_monitor_iotss_internal_sram2**

Global exclusive monitor.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `PVBusExclusiveMonitor`.

**`FVP_MPS2_Cortex_M35P.fvp_mps2.sse200.exclusive_monitor_iotss_internal_sram3`**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**`FVP_MPS2_Cortex_M35P.fvp_mps2.sse200.exclusive_squasher`**

Squashes the exclusive attribute on bus transactions.
Type: `PVBusExclusiveSquasher`.

**`FVP_MPS2_Cortex_M35P.fvp_mps2.sse200.idau_labeller`**

Type: `LabellerIdauSecurity`.

**`FVP_MPS2_Cortex_M35P.fvp_mps2.sse200.internal_msniper_tcm`**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**`FVP_MPS2_Cortex_M35P.fvp_mps2.sse200.iotss_cpuidentity`**

IoT Subsystem CPU_IDENTITY registers.
Type: `IoTSS_CPUIdentity`.

**`FVP_MPS2_Cortex_M35P.fvp_mps2.sse200.iotss_internal_sram0`**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**`FVP_MPS2_Cortex_M35P.fvp_mps2.sse200.iotss_internal_sram1`**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**`FVP_MPS2_Cortex_M35P.fvp_mps2.sse200.iotss_internal_sram2`**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**`FVP_MPS2_Cortex_M35P.fvp_mps2.sse200.iotss_internal_sram3`**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**`FVP_MPS2_Cortex_M35P.fvp_mps2.sse200.iotss_systemcontrol`**

IoT Subsystem System Control registers.
Type: `IoTSS_SystemControl`.

**`FVP_MPS2_Cortex_M35P.fvp_mps2.sse200.iotss_systeminfo`**

IoT Subsystem System Information registers.
Type: `IoTSS_SystemInfo`.

**`FVP_MPS2_Cortex_M35P.fvp_mps2.sse200.mem_switch_internal_msniper_tcm`**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**`FVP_MPS2_Cortex_M35P.fvp_mps2.sse200.mem_switch_internal_sram`**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M35P.fvp_mps2.sse200.mem_switch_internal_sram_mpc**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.sse200.mem_switch_ppu**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.sse200.mhu0**

IoT Subsystem Message Handling Unit.
Type: `IoTSS_MessageHandlingUnit`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.sse200.mhu1**

IoT Subsystem Message Handling Unit.
Type: `IoTSS_MessageHandlingUnit`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.sse200.mpc_iotss_internal_sram0**

IoT Subsystem Memory Protection Controller.
Type: `IoTSS_MemoryProtectionController`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.sse200.mpc_iotss_internal_sram0.gating_disabled_thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.sse200.mpc_iotss_internal_sram1**

IoT Subsystem Memory Protection Controller.
Type: `IoTSS_MemoryProtectionController`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.sse200.mpc_iotss_internal_sram1.gating_disabled_thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.sse200.mpc_iotss_internal_sram2**

IoT Subsystem Memory Protection Controller.
Type: `IoTSS_MemoryProtectionController`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.sse200.mpc_iotss_internal_sram2.gating_disabled_thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.sse200.mpc_iotss_internal_sram3**

IoT Subsystem Memory Protection Controller.
Type: `IoTSS_MemoryProtectionController`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.sse200.mpc_iotss_internal_sram3.gating_disabled_thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M35P.fvp_mps2.sse200.nmi_or_gate**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.sse200.nonsecure_watchdog**

ARM Watchdog Module.
Type: `CMSDK_Watchdog`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.sse200.ram0_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.sse200.ram1_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.sse200.ram2_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.sse200.ram3_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.sse200.s32k_timer**

ARM Timer Module.
Type: `CMSDK_Timer`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.sse200.s32k_timer.clk_div**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.sse200.s32k_timer.counter**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.sse200.s32k_watchdog**

ARM Watchdog Module.
Type: `CMSDK_Watchdog`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.sse200.secure_control_register_block**

MPS2 Secure Control Register Block.
Type: `MPS2_SecureCtrl`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.sse200.secure_watchdog**

ARM Watchdog Module.
Type: `CMSDK_Watchdog`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.sse200.signal_router**

Signal router.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `SignalRouter`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.sse200.sys_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.sse200.timer0**

ARM Timer Module.
Type: `CMSDK_Timer`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.sse200.timer0.clk_div**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.sse200.timer0.counter**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.sse200.timer1**

ARM Timer Module.
Type: `CMSDK_Timer`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.sse200.timer1.clk_div**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.sse200.timer1.counter**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.ssram1**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.ssram2**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.stub0**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.stub1**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M35P.fvp_mps2.stub_i2c1**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.stub_i2s**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.stub_spi0**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.stub_spi2**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.svos_dualtimer**

Type: `SVOS_DualTimer`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.svos_dualtimer.svos_dualtimer0**

ARM Dual-Timer Module.
Type: `CMSDK_DualTimer`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.svos_dualtimer.svos_dualtimer0.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.svos_dualtimer.svos_dualtimer0.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.svos_dualtimer.svos_dualtimer0.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.svos_dualtimer.svos_dualtimer0.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.svos_dualtimer.svos_dualtimer1**

ARM Dual-Timer Module.
Type: `CMSDK_DualTimer`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.svos_dualtimer.svos_dualtimer1.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.svos_dualtimer.svos_dualtimer1.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.svos_dualtimer.svos_dualtimer1.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.svos_dualtimer.svos_dualtimer1.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.svos_dualtimer.svos_dualtimer2**

ARM Dual-Timer Module.
Type: `CMSDK_DualTimer`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.svos_dualtimer.svos_dualtimer2.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.svos_dualtimer.svos_dualtimer2.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.svos_dualtimer.svos_dualtimer2.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.svos_dualtimer.svos_dualtimer2.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.svos_dualtimer.svos_dualtimer3**

ARM Dual-Timer Module.
Type: `CMSDK_DualTimer`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.svos_dualtimer.svos_dualtimer3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.svos_dualtimer.svos_dualtimer3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.svos_dualtimer.svos_dualtimer3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.svos_dualtimer.svos_dualtimer3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.switch_PSRAM_M7**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.switch_svos_dualtimer**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.telnetterminal0**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.telnetterminal1**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.telnetterminal2**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.touchscreen_interface**

MPS2 Touch Screen.
Type: `MPS2_TouchScreen`.

**FVP_MPS2_Cortex_M35P.fvp_mps2.uart_overflows_or_gate**

Or Gate.
Type: `OrGate`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

# 16.8 FVP_MPS2_Cortex-M4

List of instances in FVP_MPS2_Cortex-M4.

## FVP_MPS2_Cortex-M4 instances

**FVP_MPS2_Cortex_M4**

Type: `FVP_MPS2_Cortex_M4`.

**FVP_MPS2_Cortex_M4.armcortexm4ct**

ARM CORTEXM4 CT model.
Type: `ARM_Cortex-M4`.

**FVP_MPS2_Cortex_M4.clk25Mhz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M4.clk25khz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M4.fvp_mps2**

MPS2 DUT.
Type: `FVP_MPS2`.

**FVP_MPS2_Cortex_M4.fvp_mps2.GPIO0**

ARM PrimeCell General Purpose Input/Output.
Type: `CMSDK_GPIO`.

**FVP_MPS2_Cortex_M4.fvp_mps2.GPIO1**

ARM PrimeCell General Purpose Input/Output.
Type: `CMSDK_GPIO`.

**FVP_MPS2_Cortex_M4.fvp_mps2.GPIO2**

ARM PrimeCell General Purpose Input/Output.
Type: `CMSDK_GPIO`.

**FVP_MPS2_Cortex_M4.fvp_mps2.GPIO3**

ARM PrimeCell General Purpose Input/Output.
Type: `CMSDK_GPIO`.

**FVP_MPS2_Cortex_M4.fvp_mps2.GPIO_connection_test**

Type: `GPIO_Connection_Test`.

**FVP_MPS2_Cortex_M4.fvp_mps2.GPIO_connection_test.GPIO0_port_trans**

Type: `GPIO_Port_Transfer`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M4.fvp_mps2.GPIO_connection_test.GPIO1_port_test**

Type: `GPIO1_Connection_Test`.

**FVP_MPS2_Cortex_M4.fvp_mps2.PSRAM**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M4.fvp_mps2.PSRAM_M7**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M4.fvp_mps2.UART0**

ARM CMSDK UART Module.
Type: `CMSDK_UART`.

**FVP_MPS2_Cortex_M4.fvp_mps2.UART0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M4.fvp_mps2.UART1**

ARM CMSDK UART Module.
Type: `CMSDK_UART`.

**FVP_MPS2_Cortex_M4.fvp_mps2.UART1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M4.fvp_mps2.UART2**

ARM CMSDK UART Module.
Type: `CMSDK_UART`.

**FVP_MPS2_Cortex_M4.fvp_mps2.UART2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M4.fvp_mps2.VGA_interface**

VGA display interface between main bus and visualisation.
Type: `MPS2_VGA`.

**FVP_MPS2_Cortex_M4.fvp_mps2.ahb_ppc_iotss_expansion0**

IoT Subsystem Peripheral Protection Controller.
Type: `IoTSS_PeripheralProtectionController`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M4.fvp_mps2.ahb_ppc_iotss_expansion1**

IoT Subsystem Peripheral Protection Controller.
Type: `IoTSS_PeripheralProtectionController`.

**FVP_MPS2_Cortex_M4.fvp_mps2.apb_ppc_iotss_expansion0**

IoT Subsystem Peripheral Protection Controller.
Type: `IoTSS_PeripheralProtectionController`.

**FVP_MPS2_Cortex_M4.fvp_mps2.apb_ppc_iotss_expansion1**

IoT Subsystem Peripheral Protection Controller.
Type: `IoTSS_PeripheralProtectionController`.

**FVP_MPS2_Cortex_M4.fvp_mps2.apb_ppc_iotss_expansion2**

IoT Subsystem Peripheral Protection Controller.
Type: `IoTSS_PeripheralProtectionController`.

**FVP_MPS2_Cortex_M4.fvp_mps2.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M4.fvp_mps2.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M4.fvp_mps2.cmsdk_sysctrl**

Cortex-M Simple System Control.
Type: `CMSDK_SysCtrl`.

**FVP_MPS2_Cortex_M4.fvp_mps2.cmsdk_watchdog**

ARM Watchdog Module.
Type: `CMSDK_Watchdog`.

**FVP_MPS2_Cortex_M4.fvp_mps2.cpu_wait_or_gate_0**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_Cortex_M4.fvp_mps2.cpu_wait_or_gate_1**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_Cortex_M4.fvp_mps2.dbgen_or_gate**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_Cortex_M4.fvp_mps2.dma0**

ARM PrimeCell DMA Controller(PL080/081).
Type: `PL080_DMAC`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M4.fvp_mps2.dma0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_MPS2_Cortex_M4.fvp_mps2.dma0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_MPS2_Cortex_M4.fvp_mps2.dma0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_MPS2_Cortex_M4.fvp_mps2.dma0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M4.fvp_mps2.dma0_idau_labeller**

Type: `LabellerIdauSecurity`.

**FVP_MPS2_Cortex_M4.fvp_mps2.dma0_securitymodifier**

Type: `SecurityModifier`.

**FVP_MPS2_Cortex_M4.fvp_mps2.dma1**

ARM PrimeCell DMA Controller(PL080/081).
Type: `PL080_DMAC`.

**FVP_MPS2_Cortex_M4.fvp_mps2.dma1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_MPS2_Cortex_M4.fvp_mps2.dma1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_MPS2_Cortex_M4.fvp_mps2.dma1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_MPS2_Cortex_M4.fvp_mps2.dma1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M4.fvp_mps2.dma1_idau_labeller**

Type: `LabellerIdauSecurity`.

**FVP_MPS2_Cortex_M4.fvp_mps2.dma1_securitymodifier**

Type: `SecurityModifier`.

**FVP_MPS2_Cortex_M4.fvp_mps2.dma2**

ARM PrimeCell DMA Controller(PL080/081).
Type: `PL080_DMAC`.

**FVP_MPS2_Cortex_M4.fvp_mps2.dma2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_MPS2_Cortex_M4.fvp_mps2.dma2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_MPS2_Cortex_M4.fvp_mps2.dma2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_MPS2_Cortex_M4.fvp_mps2.dma2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M4.fvp_mps2.dma2_idau_labeller**

Type: `LabellerIdauSecurity`.

**FVP_MPS2_Cortex_M4.fvp_mps2.dma2_securitymodifier**

Type: `SecurityModifier`.

**FVP_MPS2_Cortex_M4.fvp_mps2.dma3**

ARM PrimeCell DMA Controller(PL080/081).
Type: `PL080_DMAC`.

**FVP_MPS2_Cortex_M4.fvp_mps2.dma3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_MPS2_Cortex_M4.fvp_mps2.dma3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_MPS2_Cortex_M4.fvp_mps2.dma3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_MPS2_Cortex_M4.fvp_mps2.dma3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M4.fvp_mps2.dma3_idau_labeller**

Type: `LabellerIdauSecurity`.

**FVP_MPS2_Cortex_M4.fvp_mps2.dma3_securitymodifier**

Type: `SecurityModifier`.

**FVP_MPS2_Cortex_M4.fvp_mps2.exclusive_monitor_psram**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_Cortex_M4.fvp_mps2.exclusive_monitor_psram_iotss**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M4.fvp_mps2.exclusive_monitor_switch_extra_psram_iotss**
Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M4.fvp_mps2.exclusive_monitor_switch_extra_psram_mps2**
Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M4.fvp_mps2.exclusive_monitor_zbtsram1**
Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_Cortex_M4.fvp_mps2.exclusive_monitor_zbtsram2**
Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_Cortex_M4.fvp_mps2.exclusive_squasher**
Squashes the exclusive attribute on bus transactions.
Type: `PVBusExclusiveSquasher`.

**FVP_MPS2_Cortex_M4.fvp_mps2.fpga_sysctrl**
FPGA SysCtrl timers LEDs and switches.
Type: `FPGA_SysCtrl`.

**FVP_MPS2_Cortex_M4.fvp_mps2.fpga_sysctrl.callBack100HzCounter**
Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M4.fvp_mps2.fpga_sysctrl.clockdivider**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M4.fvp_mps2.gpio_0_or_2**
Or Gate.
Type: `OrGate`.

**FVP_MPS2_Cortex_M4.fvp_mps2.gpio_1_or_3**
Or Gate.
Type: `OrGate`.

**FVP_MPS2_Cortex_M4.fvp_mps2.hostbridge**
Host Socket Interface Component.
Type: `HostBridge`.

**FVP_MPS2_Cortex_M4.fvp_mps2.mem_switch_extra_psram_iotss**
Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M4.fvp_mps2.mem_switch_extra_psram_mps2**
Allow transactions to be routed arbitrarily.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M4.fvp_mps2.mpc_iotss_ssram1**

IoT Subsystem Memory Protection Controller.
Type: `IoTSS_MemoryProtectionController`.

**FVP_MPS2_Cortex_M4.fvp_mps2.mpc_iotss_ssram1.gating_disabled_thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M4.fvp_mps2.mpc_iotss_ssram2**

IoT Subsystem Memory Protection Controller.
Type: `IoTSS_MemoryProtectionController`.

**FVP_MPS2_Cortex_M4.fvp_mps2.mpc_iotss_ssram2.gating_disabled_thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M4.fvp_mps2.mpc_iotss_ssram3**

IoT Subsystem Memory Protection Controller.
Type: `IoTSS_MemoryProtectionController`.

**FVP_MPS2_Cortex_M4.fvp_mps2.mpc_iotss_ssram3.gating_disabled_thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M4.fvp_mps2.mps2_audio**

MPS2 Audio.
Type: `MPS2_Audio`.

**FVP_MPS2_Cortex_M4.fvp_mps2.mps2_cmsdk_dualtimer**

ARM Dual-Timer Module.
Type: `CMSDK_DualTimer`.

**FVP_MPS2_Cortex_M4.fvp_mps2.mps2_cmsdk_dualtimer.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M4.fvp_mps2.mps2_cmsdk_dualtimer.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M4.fvp_mps2.mps2_cmsdk_dualtimer.counter0**

Internal component used by SP804 Timer module.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `CounterModule`.

**FVP_MPS2_Cortex_M4.fvp_mps2.mps2_cmsdk_dualtimer.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M4.fvp_mps2.mps2_exclusive_monitor_zbtsram1**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_Cortex_M4.fvp_mps2.mps2_exclusive_monitor_zbtsram2**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_Cortex_M4.fvp_mps2.mps2_lcd**

MPS2 LCD I2C interface.
Type: `MPS2_LCD`.

**FVP_MPS2_Cortex_M4.fvp_mps2.mps2_mpc_iotss_ssram1**

IoT Subsystem Memory Protection Controller.
Type: `IoTSS_MemoryProtectionController`.

**FVP_MPS2_Cortex_M4.fvp_mps2.mps2_mpc_iotss_ssram1.gating_disabled_thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M4.fvp_mps2.mps2_secure_control_register_block**

MPS2 Secure Control Register Block.
Type: `MPS2_SecureCtrl`.

**FVP_MPS2_Cortex_M4.fvp_mps2.mps2_timer0**

ARM Timer Module.
Type: `CMSDK_Timer`.

**FVP_MPS2_Cortex_M4.fvp_mps2.mps2_timer0.clk_div**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M4.fvp_mps2.mps2_timer0.counter**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M4.fvp_mps2.mps2_timer1**

ARM Timer Module.
Type: `CMSDK_Timer`.

**FVP_MPS2_Cortex_M4.fvp_mps2.mps2_timer1.clk_div**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M4.fvp_mps2.mps2_timer1.counter**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M4.fvp_mps2.mps2_visualisation**

Display window for MPS2 using Visualisation library.
Type: `MPS2_Visualisation`.

**FVP_MPS2_Cortex_M4.fvp_mps2.niden_or_gate**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_Cortex_M4.fvp_mps2.nmi_or_gate**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_Cortex_M4.fvp_mps2.pl022_ssp_mps2**

ARM PrimeCell Synchronous Serial Port(PL022).
Type: `PL022_SSP_MPS2`.

**FVP_MPS2_Cortex_M4.fvp_mps2.pl022_ssp_mps2.prescaler**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M4.fvp_mps2.platform_bus_switch**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M4.fvp_mps2.platform_switch_dma0**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M4.fvp_mps2.platform_switch_dma1**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M4.fvp_mps2.signal_router**

Signal router.
Type: `SignalRouter`.

**FVP_MPS2_Cortex_M4.fvp_mps2.smsc_91c111**

SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_MPS2_Cortex_M4.fvp_mps2.spiden_or_gate**

Or Gate.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `OrGate`.

**FVP_MPS2_Cortex_M4.fvp_mps2.spniden_or_gate**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_Cortex_M4.fvp_mps2.sse200**

SSE-200 subsystem.
Type: `SSE200`.

**FVP_MPS2_Cortex_M4.fvp_mps2.sse200.acg_cpu0**

IoT Subsystem Access Control Gate.
Type: `IoTSS_AccessControlGate`.

**FVP_MPS2_Cortex_M4.fvp_mps2.sse200.acg_cpu1**

IoT Subsystem Access Control Gate.
Type: `IoTSS_AccessControlGate`.

**FVP_MPS2_Cortex_M4.fvp_mps2.sse200.acg_sram0**

IoT Subsystem Access Control Gate.
Type: `IoTSS_AccessControlGate`.

**FVP_MPS2_Cortex_M4.fvp_mps2.sse200.acg_sram1**

IoT Subsystem Access Control Gate.
Type: `IoTSS_AccessControlGate`.

**FVP_MPS2_Cortex_M4.fvp_mps2.sse200.acg_sram2**

IoT Subsystem Access Control Gate.
Type: `IoTSS_AccessControlGate`.

**FVP_MPS2_Cortex_M4.fvp_mps2.sse200.acg_sram3**

IoT Subsystem Access Control Gate.
Type: `IoTSS_AccessControlGate`.

**FVP_MPS2_Cortex_M4.fvp_mps2.sse200.apb_ppc_iotss_subsystem0**

IoT Subsystem Peripheral Protection Controller.
Type: `IoTSS_PeripheralProtectionController`.

**FVP_MPS2_Cortex_M4.fvp_mps2.sse200.apb_ppc_iotss_subsystem1**

IoT Subsystem Peripheral Protection Controller.
Type: `IoTSS_PeripheralProtectionController`.

**FVP_MPS2_Cortex_M4.fvp_mps2.sse200.clock32kHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M4.fvp_mps2.sse200.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M4.fvp_mps2.sse200.cmsdk_dualtimer**

ARM Dual-Timer Module.
Type: `CMSDK_DualTimer`.

**FVP_MPS2_Cortex_M4.fvp_mps2.sse200.cmsdk_dualtimer.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M4.fvp_mps2.sse200.cmsdk_dualtimer.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M4.fvp_mps2.sse200.cmsdk_dualtimer.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M4.fvp_mps2.sse200.cmsdk_dualtimer.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M4.fvp_mps2.sse200.cordio_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M4.fvp_mps2.sse200.cpu0core_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M4.fvp_mps2.sse200.cpu0dbg_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M4.fvp_mps2.sse200.cpu1core_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M4.fvp_mps2.sse200.cpu1dbg_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M4.fvp_mps2.sse200.crypto_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M4.fvp_mps2.sse200.dbg_ppu**
ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M4.fvp_mps2.sse200.exclusive_monitor_iotss_internal_sram0**
Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_Cortex_M4.fvp_mps2.sse200.exclusive_monitor_iotss_internal_sram1**
Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_Cortex_M4.fvp_mps2.sse200.exclusive_monitor_iotss_internal_sram2**
Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_Cortex_M4.fvp_mps2.sse200.exclusive_monitor_iotss_internal_sram3**
Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_Cortex_M4.fvp_mps2.sse200.exclusive_squasher**
Squashes the exclusive attribute on bus transactions.
Type: `PVBusExclusiveSquasher`.

**FVP_MPS2_Cortex_M4.fvp_mps2.sse200.idau_labeller**
Type: `LabellerIdauSecurity`.

**FVP_MPS2_Cortex_M4.fvp_mps2.sse200.internal_msniper_tcm**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M4.fvp_mps2.sse200.iotss_cpuidentity**
IoT Subsystem CPU_IDENTITY registers.
Type: `IoTSS_CPUIdentity`.

**FVP_MPS2_Cortex_M4.fvp_mps2.sse200.iotss_internal_sram0**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M4.fvp_mps2.sse200.iotss_internal_sram1**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M4.fvp_mps2.sse200.iotss_internal_sram2**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M4.fvp_mps2.sse200.iotss_internal_sram3**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M4.fvp_mps2.sse200.iotss_systemcontrol**
IoT Subsystem System Control registers.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `IoTSS_SystemControl`.

**FVP_MPS2_Cortex_M4.fvp_mps2.sse200.iotss_systeminfo**

IoT Subsystem System Information registers.
Type: `IoTSS_SystemInfo`.

**FVP_MPS2_Cortex_M4.fvp_mps2.sse200.mem_switch_internal_msniper_tcm**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M4.fvp_mps2.sse200.mem_switch_internal_sram**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M4.fvp_mps2.sse200.mem_switch_internal_sram_mpc**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M4.fvp_mps2.sse200.mem_switch_ppu**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M4.fvp_mps2.sse200.mhu0**

IoT Subsystem Message Handling Unit.
Type: `IoTSS_MessageHandlingUnit`.

**FVP_MPS2_Cortex_M4.fvp_mps2.sse200.mhu1**

IoT Subsystem Message Handling Unit.
Type: `IoTSS_MessageHandlingUnit`.

**FVP_MPS2_Cortex_M4.fvp_mps2.sse200.mpc_iotss_internal_sram0**

IoT Subsystem Memory Protection Controller.
Type: `IoTSS_MemoryProtectionController`.

**FVP_MPS2_Cortex_M4.fvp_mps2.sse200.mpc_iotss_internal_sram0.gating_disabled_thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M4.fvp_mps2.sse200.mpc_iotss_internal_sram1**

IoT Subsystem Memory Protection Controller.
Type: `IoTSS_MemoryProtectionController`.

**FVP_MPS2_Cortex_M4.fvp_mps2.sse200.mpc_iotss_internal_sram1.gating_disabled_thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M4.fvp_mps2.sse200.mpc_iotss_internal_sram2**

IoT Subsystem Memory Protection Controller.
Type: `IoTSS_MemoryProtectionController`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M4.fvp_mps2.sse200.mpc_iotss_internal_sram2.gating_disabled_thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M4.fvp_mps2.sse200.mpc_iotss_internal_sram3**

IoT Subsystem Memory Protection Controller.
Type: `IoTSS_MemoryProtectionController`.

**FVP_MPS2_Cortex_M4.fvp_mps2.sse200.mpc_iotss_internal_sram3.gating_disabled_thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M4.fvp_mps2.sse200.nmi_or_gate**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_Cortex_M4.fvp_mps2.sse200.nonsecure_watchdog**

ARM Watchdog Module.
Type: `CMSDK_Watchdog`.

**FVP_MPS2_Cortex_M4.fvp_mps2.sse200.ram0_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M4.fvp_mps2.sse200.ram1_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M4.fvp_mps2.sse200.ram2_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M4.fvp_mps2.sse200.ram3_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M4.fvp_mps2.sse200.s32k_timer**

ARM Timer Module.
Type: `CMSDK_Timer`.

**FVP_MPS2_Cortex_M4.fvp_mps2.sse200.s32k_timer.clk_div**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M4.fvp_mps2.sse200.s32k_timer.counter**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M4.fvp_mps2.sse200.s32k_watchdog**

ARM Watchdog Module.
Type: `CMSDK_Watchdog`.

**FVP_MPS2_Cortex_M4.fvp_mps2.sse200.secure_control_register_block**

MPS2 Secure Control Register Block.
Type: `MPS2_SecureCtrl`.

**FVP_MPS2_Cortex_M4.fvp_mps2.sse200.secure_watchdog**

ARM Watchdog Module.
Type: `CMSDK_Watchdog`.

**FVP_MPS2_Cortex_M4.fvp_mps2.sse200.signal_router**

Signal router.
Type: `SignalRouter`.

**FVP_MPS2_Cortex_M4.fvp_mps2.sse200.sys_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M4.fvp_mps2.sse200.timer0**

ARM Timer Module.
Type: `CMSDK_Timer`.

**FVP_MPS2_Cortex_M4.fvp_mps2.sse200.timer0.clk_div**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M4.fvp_mps2.sse200.timer0.counter**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M4.fvp_mps2.sse200.timer1**

ARM Timer Module.
Type: `CMSDK_Timer`.

**FVP_MPS2_Cortex_M4.fvp_mps2.sse200.timer1.clk_div**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M4.fvp_mps2.sse200.timer1.counter**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M4.fvp_mps2.ssram1**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M4.fvp_mps2.ssram2**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M4.fvp_mps2.stub0**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M4.fvp_mps2.stub1**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M4.fvp_mps2.stub_i2c1**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M4.fvp_mps2.stub_i2s**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M4.fvp_mps2.stub_spi0**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M4.fvp_mps2.stub_spi2**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M4.fvp_mps2.svos_dualtimer**

Type: `SVOS_DualTimer`.

**FVP_MPS2_Cortex_M4.fvp_mps2.svos_dualtimer.svos_dualtimer0**

ARM Dual-Timer Module.
Type: `CMSDK_DualTimer`.

**FVP_MPS2_Cortex_M4.fvp_mps2.svos_dualtimer.svos_dualtimer0.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M4.fvp_mps2.svos_dualtimer.svos_dualtimer0.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M4.fvp_mps2.svos_dualtimer.svos_dualtimer0.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M4.fvp_mps2.svos_dualtimer.svos_dualtimer0.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M4.fvp_mps2.svos_dualtimer.svos_dualtimer1**

ARM Dual-Timer Module.
Type: `CMSDK_DualTimer`.

**FVP_MPS2_Cortex_M4.fvp_mps2.svos_dualtimer.svos_dualtimer1.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M4.fvp_mps2.svos_dualtimer.svos_dualtimer1.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M4.fvp_mps2.svos_dualtimer.svos_dualtimer1.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M4.fvp_mps2.svos_dualtimer.svos_dualtimer1.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M4.fvp_mps2.svos_dualtimer.svos_dualtimer2**

ARM Dual-Timer Module.
Type: `CMSDK_DualTimer`.

**FVP_MPS2_Cortex_M4.fvp_mps2.svos_dualtimer.svos_dualtimer2.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M4.fvp_mps2.svos_dualtimer.svos_dualtimer2.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M4.fvp_mps2.svos_dualtimer.svos_dualtimer2.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M4.fvp_mps2.svos_dualtimer.svos_dualtimer2.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M4.fvp_mps2.svos_dualtimer.svos_dualtimer3**

ARM Dual-Timer Module.
Type: `CMSDK_DualTimer`.

**FVP_MPS2_Cortex_M4.fvp_mps2.svos_dualtimer.svos_dualtimer3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M4.fvp_mps2.svos_dualtimer.svos_dualtimer3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M4.fvp_mps2.svos_dualtimer.svos_dualtimer3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M4.fvp_mps2.svos_dualtimer.svos_dualtimer3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M4.fvp_mps2.switch_PSRAM_M7**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M4.fvp_mps2.switch_svos_dualtimer**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M4.fvp_mps2.telnetterminal0**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_MPS2_Cortex_M4.fvp_mps2.telnetterminal1**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_MPS2_Cortex_M4.fvp_mps2.telnetterminal2**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_MPS2_Cortex_M4.fvp_mps2.touchscreen_interface**

MPS2 Touch Screen.
Type: `MPS2_TouchScreen`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M4.fvp_mps2.uart_overflows_or_gate**

Or Gate.
Type: `OrGate`.

# 16.9 FVP_MPS2_Cortex-M55

List of instances in FVP_MPS2_Cortex-M55.

## Differences between the FVP and the MPS2 specification

- The model does not have the random number generator or unique ID/secure storage mentioned in the MPS2 specification because the programmer's view of these devices is yet to be defined.
- The Ethernet controller in the model is a LAN91C111. The MPS2 documents, including those for pre-v8-M cores, specify a LAN9220.
- MTB, ETM, and TPIU are not supported. MTB RAM is absent.
- In the Memory Gating Unit, the model provides a configurable block size. For performance reasons, the minimum block size in the model is 4096 bytes. Hardware and later models might allow smaller block sizes. Software should always use the `BLK_CFG` register to determine block size.
- As in other MPS2 Fast Models, some of the peripherals have minimal implementations:
  - The Audio controller is RAZ/WI.
  - Only the touchscreen functionality of the STMPE811 touchscreen controller is implemented.
  - A subset of the Ampire LCD module's graphics modes are supported.

## FVP_MPS2_Cortex-M55 instances

**FVP_MPS2_Cortex_M55**

Type: `FVP_MPS2_Cortex_M55`.

**FVP_MPS2_Cortex_M55.clk25Mhz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M55.clk25khz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M55.core0_bus_gasket**
writes are ignored, non-word reads abort, and word reads take values from that fifo.
Type: STLBusGasket.

**FVP_MPS2_Cortex_M55.cpu0**
ARM Cortex-M55 CT model.
Type: ARM_Cortex-M55.

**FVP_MPS2_Cortex_M55.cpu1**
ARM Cortex-M55 CT model.
Type: ARM_Cortex-M55.

**FVP_MPS2_Cortex_M55.fvp_mps2**
MPS2 DUT.
Type: FVP_MPS2.

**FVP_MPS2_Cortex_M55.fvp_mps2.GPIO0**
ARM PrimeCell General Purpose Input/Output.
Type: CMSDK_GPIO.

**FVP_MPS2_Cortex_M55.fvp_mps2.GPIO1**
ARM PrimeCell General Purpose Input/Output.
Type: CMSDK_GPIO.

**FVP_MPS2_Cortex_M55.fvp_mps2.GPIO2**
ARM PrimeCell General Purpose Input/Output.
Type: CMSDK_GPIO.

**FVP_MPS2_Cortex_M55.fvp_mps2.GPIO3**
ARM PrimeCell General Purpose Input/Output.
Type: CMSDK_GPIO.

**FVP_MPS2_Cortex_M55.fvp_mps2.GPIO_connection_test**
Type: GPIO_Connection_Test.

**FVP_MPS2_Cortex_M55.fvp_mps2.GPIO_connection_test.GPIO0_port_trans**
Type: GPIO_Port_Transfer.

**FVP_MPS2_Cortex_M55.fvp_mps2.GPIO_connection_test.GPIO1_port_test**
Type: GPIO1_Connection_Test.

**FVP_MPS2_Cortex_M55.fvp_mps2.PSRAM**
RAM device, can be dynamic or static ram.
Type: RAMDevice.

**FVP_MPS2_Cortex_M55.fvp_mps2.PSRAM_M7**
RAM device, can be dynamic or static ram.
Type: RAMDevice.

**FVP_MPS2_Cortex_M55.fvp_mps2.UART0**
ARM CMSDK UART Module.
Type: CMSDK_UART.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M55.fvp_mps2.UART0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M55.fvp_mps2.UART1**

ARM CMSDK UART Module.
Type: `CMSDK_UART`.

**FVP_MPS2_Cortex_M55.fvp_mps2.UART1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M55.fvp_mps2.UART2**

ARM CMSDK UART Module.
Type: `CMSDK_UART`.

**FVP_MPS2_Cortex_M55.fvp_mps2.UART2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M55.fvp_mps2.VGA_interface**

VGA display interface between main bus and visualisation.
Type: `MPS2_VGA`.

**FVP_MPS2_Cortex_M55.fvp_mps2.ahb_ppc_iotss_expansion0**

IoT Subsystem Peripheral Protection Controller.
Type: `IoTSS_PeripheralProtectionController`.

**FVP_MPS2_Cortex_M55.fvp_mps2.ahb_ppc_iotss_expansion1**

IoT Subsystem Peripheral Protection Controller.
Type: `IoTSS_PeripheralProtectionController`.

**FVP_MPS2_Cortex_M55.fvp_mps2.apb_ppc_iotss_expansion0**

IoT Subsystem Peripheral Protection Controller.
Type: `IoTSS_PeripheralProtectionController`.

**FVP_MPS2_Cortex_M55.fvp_mps2.apb_ppc_iotss_expansion1**

IoT Subsystem Peripheral Protection Controller.
Type: `IoTSS_PeripheralProtectionController`.

**FVP_MPS2_Cortex_M55.fvp_mps2.apb_ppc_iotss_expansion2**

IoT Subsystem Peripheral Protection Controller.
Type: `IoTSS_PeripheralProtectionController`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M55.fvp_mps2.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M55.fvp_mps2.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M55.fvp_mps2.cmsdk_sysctrl**

Cortex-M Simple System Control.
Type: `CMSDK_SysCtrl`.

**FVP_MPS2_Cortex_M55.fvp_mps2.cmsdk_watchdog**

ARM Watchdog Module.
Type: `CMSDK_Watchdog`.

**FVP_MPS2_Cortex_M55.fvp_mps2.cpu_wait_or_gate_0**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_Cortex_M55.fvp_mps2.cpu_wait_or_gate_1**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_Cortex_M55.fvp_mps2.dbgen_or_gate**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_Cortex_M55.fvp_mps2.dma0**

ARM PrimeCell DMA Controller(PL080/081).
Type: `PL080_DMAC`.

**FVP_MPS2_Cortex_M55.fvp_mps2.dma0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_MPS2_Cortex_M55.fvp_mps2.dma0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_MPS2_Cortex_M55.fvp_mps2.dma0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_MPS2_Cortex_M55.fvp_mps2.dma0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M55.fvp_mps2.dma0_idau_labeller**

Type: `LabellerIdauSecurity`.

**FVP_MPS2_Cortex_M55.fvp_mps2.dma0_securitymodifier**

Type: `SecurityModifier`.

**FVP_MPS2_Cortex_M55.fvp_mps2.dma1**

ARM PrimeCell DMA Controller(PL080/081).
Type: `PL080_DMAC`.

**FVP_MPS2_Cortex_M55.fvp_mps2.dma1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_MPS2_Cortex_M55.fvp_mps2.dma1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_MPS2_Cortex_M55.fvp_mps2.dma1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_MPS2_Cortex_M55.fvp_mps2.dma1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M55.fvp_mps2.dma1_idau_labeller**

Type: `LabellerIdauSecurity`.

**FVP_MPS2_Cortex_M55.fvp_mps2.dma1_securitymodifier**

Type: `SecurityModifier`.

**FVP_MPS2_Cortex_M55.fvp_mps2.dma2**

ARM PrimeCell DMA Controller(PL080/081).
Type: `PL080_DMAC`.

**FVP_MPS2_Cortex_M55.fvp_mps2.dma2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_MPS2_Cortex_M55.fvp_mps2.dma2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_MPS2_Cortex_M55.fvp_mps2.dma2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_MPS2_Cortex_M55.fvp_mps2.dma2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M55.fvp_mps2.dma2_idau_labeller**

Type: `LabellerIdauSecurity`.

**FVP_MPS2_Cortex_M55.fvp_mps2.dma2_securitymodifier**

Type: `SecurityModifier`.

**FVP_MPS2_Cortex_M55.fvp_mps2.dma3**

ARM PrimeCell DMA Controller(PL080/081).
Type: `PL080_DMAC`.

**FVP_MPS2_Cortex_M55.fvp_mps2.dma3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_MPS2_Cortex_M55.fvp_mps2.dma3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_MPS2_Cortex_M55.fvp_mps2.dma3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_MPS2_Cortex_M55.fvp_mps2.dma3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M55.fvp_mps2.dma3_idau_labeller**

Type: `LabellerIdauSecurity`.

**FVP_MPS2_Cortex_M55.fvp_mps2.dma3_securitymodifier**

Type: `SecurityModifier`.

**FVP_MPS2_Cortex_M55.fvp_mps2.exclusive_monitor_psram**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_Cortex_M55.fvp_mps2.exclusive_monitor_psram_iotss**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_Cortex_M55.fvp_mps2.exclusive_monitor_switch_extra_psram_iotss**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M55.fvp_mps2.exclusive_monitor_switch_extra_psram_mps2**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M55.fvp_mps2.exclusive_monitor_zbtsram1**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_Cortex_M55.fvp_mps2.exclusive_monitor_zbtsram2**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M55.fvp_mps2.exclusive_squasher**

Squashes the exclusive attribute on bus transactions.
Type: `PVBusExclusiveSquasher`.

**FVP_MPS2_Cortex_M55.fvp_mps2.fpga_sysctrl**

FPGA SysCtrl timers LEDs and switches.
Type: `FPGA_SysCtrl`.

**FVP_MPS2_Cortex_M55.fvp_mps2.fpga_sysctrl.callBack100HzCounter**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M55.fvp_mps2.fpga_sysctrl.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M55.fvp_mps2.gpio_0_or_2**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_Cortex_M55.fvp_mps2.gpio_1_or_3**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_Cortex_M55.fvp_mps2.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_MPS2_Cortex_M55.fvp_mps2.mem_switch_extra_psram_iotss**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M55.fvp_mps2.mem_switch_extra_psram_mps2**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M55.fvp_mps2.mpc_iotss_ssram1**

IoT Subsystem Memory Protection Controller.
Type: `IoTSS_MemoryProtectionController`.

**FVP_MPS2_Cortex_M55.fvp_mps2.mpc_iotss_ssram1.gating_disabled_thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M55.fvp_mps2.mpc_iotss_ssram2**

IoT Subsystem Memory Protection Controller.
Type: `IoTSS_MemoryProtectionController`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M55.fvp_mps2.mpc_iotss_ssram2.gating_disabled_thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M55.fvp_mps2.mpc_iotss_ssram3**

IoT Subsystem Memory Protection Controller.
Type: `IoTSS_MemoryProtectionController`.

**FVP_MPS2_Cortex_M55.fvp_mps2.mpc_iotss_ssram3.gating_disabled_thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M55.fvp_mps2.mps2_audio**

MPS2 Audio.
Type: `MPS2_Audio`.

**FVP_MPS2_Cortex_M55.fvp_mps2.mps2_cmsdk_dualtimer**

ARM Dual-Timer Module.
Type: `CMSDK_DualTimer`.

**FVP_MPS2_Cortex_M55.fvp_mps2.mps2_cmsdk_dualtimer.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M55.fvp_mps2.mps2_cmsdk_dualtimer.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M55.fvp_mps2.mps2_cmsdk_dualtimer.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M55.fvp_mps2.mps2_cmsdk_dualtimer.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M55.fvp_mps2.mps2_exclusive_monitor_zbtsram1**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_Cortex_M55.fvp_mps2.mps2_exclusive_monitor_zbtsram2**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M55.fvp_mps2.mps2_lcd**

MPS2 LCD I2C interface.
Type: `MPS2_LCD`.

**FVP_MPS2_Cortex_M55.fvp_mps2.mps2_mpc_iotss_ssram1**

IoT Subsystem Memory Protection Controller.
Type: `IoTSS_MemoryProtectionController`.

**FVP_MPS2_Cortex_M55.fvp_mps2.mps2_mpc_iotss_ssram1.gating_disabled_thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M55.fvp_mps2.mps2_secure_control_register_block**

MPS2 Secure Control Register Block.
Type: `MPS2_SecureCtrl`.

**FVP_MPS2_Cortex_M55.fvp_mps2.mps2_timer0**

ARM Timer Module.
Type: `CMSDK_Timer`.

**FVP_MPS2_Cortex_M55.fvp_mps2.mps2_timer0.clk_div**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M55.fvp_mps2.mps2_timer0.counter**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M55.fvp_mps2.mps2_timer1**

ARM Timer Module.
Type: `CMSDK_Timer`.

**FVP_MPS2_Cortex_M55.fvp_mps2.mps2_timer1.clk_div**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M55.fvp_mps2.mps2_timer1.counter**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M55.fvp_mps2.mps2_visualisation**

Display window for MPS2 using Visualisation library.
Type: `MPS2_Visualisation`.

**FVP_MPS2_Cortex_M55.fvp_mps2.niden_or_gate**

Or Gate.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `OrGate`.

**FVP_MPS2_Cortex_M55.fvp_mps2.nmi_or_gate**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_Cortex_M55.fvp_mps2.pl022_ssp_mps2**

ARM PrimeCell Synchronous Serial Port(PL022).
Type: `PL022_SSP_MPS2`.

**FVP_MPS2_Cortex_M55.fvp_mps2.pl022_ssp_mps2.prescaler**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M55.fvp_mps2.platform_bus_switch**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M55.fvp_mps2.platform_switch_dma0**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M55.fvp_mps2.platform_switch_dma1**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M55.fvp_mps2.signal_router**

Signal router.
Type: `SignalRouter`.

**FVP_MPS2_Cortex_M55.fvp_mps2.smsc_91c111**

SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_MPS2_Cortex_M55.fvp_mps2.spiden_or_gate**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_Cortex_M55.fvp_mps2.spniden_or_gate**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_Cortex_M55.fvp_mps2.sse200**

SSE-200 subsystem.
Type: `SSE200`.

**FVP_MPS2_Cortex_M55.fvp_mps2.sse200.acg_cpu0**

IoT Subsystem Access Control Gate.
Type: `IoTSS_AccessControlGate`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M55.fvp_mps2.sse200.acg_cpu1**
IoT Subsystem Access Control Gate.
Type: `IoTSS_AccessControlGate`.

**FVP_MPS2_Cortex_M55.fvp_mps2.sse200.acg_sram0**
IoT Subsystem Access Control Gate.
Type: `IoTSS_AccessControlGate`.

**FVP_MPS2_Cortex_M55.fvp_mps2.sse200.acg_sram1**
IoT Subsystem Access Control Gate.
Type: `IoTSS_AccessControlGate`.

**FVP_MPS2_Cortex_M55.fvp_mps2.sse200.acg_sram2**
IoT Subsystem Access Control Gate.
Type: `IoTSS_AccessControlGate`.

**FVP_MPS2_Cortex_M55.fvp_mps2.sse200.acg_sram3**
IoT Subsystem Access Control Gate.
Type: `IoTSS_AccessControlGate`.

**FVP_MPS2_Cortex_M55.fvp_mps2.sse200.apb_ppc_iotss_subsystem0**
IoT Subsystem Peripheral Protection Controller.
Type: `IoTSS_PeripheralProtectionController`.

**FVP_MPS2_Cortex_M55.fvp_mps2.sse200.apb_ppc_iotss_subsystem1**
IoT Subsystem Peripheral Protection Controller.
Type: `IoTSS_PeripheralProtectionController`.

**FVP_MPS2_Cortex_M55.fvp_mps2.sse200.clock32kHz**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M55.fvp_mps2.sse200.clockdivider**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M55.fvp_mps2.sse200.cmsdk_dualtimer**
ARM Dual-Timer Module.
Type: `CMSDK_DualTimer`.

**FVP_MPS2_Cortex_M55.fvp_mps2.sse200.cmsdk_dualtimer.clk_div0**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M55.fvp_mps2.sse200.cmsdk_dualtimer.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M55.fvp_mps2.sse200.cmsdk_dualtimer.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M55.fvp_mps2.sse200.cmsdk_dualtimer.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M55.fvp_mps2.sse200.cordio_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M55.fvp_mps2.sse200.cpu0core_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M55.fvp_mps2.sse200.cpu0dbg_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M55.fvp_mps2.sse200.cpu1core_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M55.fvp_mps2.sse200.cpu1dbg_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M55.fvp_mps2.sse200.crypto_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M55.fvp_mps2.sse200.dbg_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M55.fvp_mps2.sse200.exclusive_monitor_iotss_internal_sram0**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_Cortex_M55.fvp_mps2.sse200.exclusive_monitor_iotss_internal_sram1**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_Cortex_M55.fvp_mps2.sse200.exclusive_monitor_iotss_internal_sram2**

Global exclusive monitor.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_Cortex_M55.fvp_mps2.sse200.exclusive_monitor_iotss_internal_sram3**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_Cortex_M55.fvp_mps2.sse200.exclusive_squasher**

Squashes the exclusive attribute on bus transactions.
Type: `PVBusExclusiveSquasher`.

**FVP_MPS2_Cortex_M55.fvp_mps2.sse200.idau_labeller**

Type: `LabellerIdauSecurity`.

**FVP_MPS2_Cortex_M55.fvp_mps2.sse200.internal_msniper_tcm**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M55.fvp_mps2.sse200.iotss_cpuidentity**

IoT Subsystem CPU_IDENTITY registers.
Type: `IoTSS_CPUIdentity`.

**FVP_MPS2_Cortex_M55.fvp_mps2.sse200.iotss_internal_sram0**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M55.fvp_mps2.sse200.iotss_internal_sram1**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M55.fvp_mps2.sse200.iotss_internal_sram2**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M55.fvp_mps2.sse200.iotss_internal_sram3**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M55.fvp_mps2.sse200.iotss_systemcontrol**

IoT Subsystem System Control registers.
Type: `IoTSS_SystemControl`.

**FVP_MPS2_Cortex_M55.fvp_mps2.sse200.iotss_systeminfo**

IoT Subsystem System Information registers.
Type: `IoTSS_SystemInfo`.

**FVP_MPS2_Cortex_M55.fvp_mps2.sse200.mem_switch_internal_msniper_tcm**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M55.fvp_mps2.sse200.mem_switch_internal_sram**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M55.fvp_mps2.sse200.mem_switch_internal_sram_mpc**
Allow transactions to be routed arbitrarily.
Type: PVBusRouter.

**FVP_MPS2_Cortex_M55.fvp_mps2.sse200.mem_switch_ppu**
Allow transactions to be routed arbitrarily.
Type: PVBusRouter.

**FVP_MPS2_Cortex_M55.fvp_mps2.sse200.mhu0**
IoT Subsystem Message Handling Unit.
Type: IoTSS_MessageHandlingUnit.

**FVP_MPS2_Cortex_M55.fvp_mps2.sse200.mhu1**
IoT Subsystem Message Handling Unit.
Type: IoTSS_MessageHandlingUnit.

**FVP_MPS2_Cortex_M55.fvp_mps2.sse200.mpc_iotss_internal_sram0**
IoT Subsystem Memory Protection Controller.
Type: IoTSS_MemoryProtectionController.

**FVP_MPS2_Cortex_M55.fvp_mps2.sse200.mpc_iotss_internal_sram0.gating_disabled_thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: SchedulerThreadEvent.

**FVP_MPS2_Cortex_M55.fvp_mps2.sse200.mpc_iotss_internal_sram1**
IoT Subsystem Memory Protection Controller.
Type: IoTSS_MemoryProtectionController.

**FVP_MPS2_Cortex_M55.fvp_mps2.sse200.mpc_iotss_internal_sram1.gating_disabled_thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: SchedulerThreadEvent.

**FVP_MPS2_Cortex_M55.fvp_mps2.sse200.mpc_iotss_internal_sram2**
IoT Subsystem Memory Protection Controller.
Type: IoTSS_MemoryProtectionController.

**FVP_MPS2_Cortex_M55.fvp_mps2.sse200.mpc_iotss_internal_sram2.gating_disabled_thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: SchedulerThreadEvent.

**FVP_MPS2_Cortex_M55.fvp_mps2.sse200.mpc_iotss_internal_sram3**
IoT Subsystem Memory Protection Controller.
Type: IoTSS_MemoryProtectionController.

**FVP_MPS2_Cortex_M55.fvp_mps2.sse200.mpc_iotss_internal_sram3.gating_disabled_thread_event**
A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: SchedulerThreadEvent.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M55.fvp_mps2.sse200.nmi_or_gate**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_Cortex_M55.fvp_mps2.sse200.nonsecure_watchdog**

ARM Watchdog Module.
Type: `CMSDK_Watchdog`.

**FVP_MPS2_Cortex_M55.fvp_mps2.sse200.ram0_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M55.fvp_mps2.sse200.ram1_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M55.fvp_mps2.sse200.ram2_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M55.fvp_mps2.sse200.ram3_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M55.fvp_mps2.sse200.s32k_timer**

ARM Timer Module.
Type: `CMSDK_Timer`.

**FVP_MPS2_Cortex_M55.fvp_mps2.sse200.s32k_timer.clk_div**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M55.fvp_mps2.sse200.s32k_timer.counter**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M55.fvp_mps2.sse200.s32k_watchdog**

ARM Watchdog Module.
Type: `CMSDK_Watchdog`.

**FVP_MPS2_Cortex_M55.fvp_mps2.sse200.secure_control_register_block**

MPS2 Secure Control Register Block.
Type: `MPS2_SecureCtrl`.

**FVP_MPS2_Cortex_M55.fvp_mps2.sse200.secure_watchdog**

ARM Watchdog Module.
Type: `CMSDK_Watchdog`.

**FVP_MPS2_Cortex_M55.fvp_mps2.sse200.signal_router**

Signal router.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `SignalRouter`.

**FVP_MPS2_Cortex_M55.fvp_mps2.sse200.sys_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M55.fvp_mps2.sse200.timer0**

ARM Timer Module.
Type: `CMSDK_Timer`.

**FVP_MPS2_Cortex_M55.fvp_mps2.sse200.timer0.clk_div**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M55.fvp_mps2.sse200.timer0.counter**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M55.fvp_mps2.sse200.timer1**

ARM Timer Module.
Type: `CMSDK_Timer`.

**FVP_MPS2_Cortex_M55.fvp_mps2.sse200.timer1.clk_div**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M55.fvp_mps2.sse200.timer1.counter**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M55.fvp_mps2.ssram1**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M55.fvp_mps2.ssram2**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M55.fvp_mps2.stub0**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M55.fvp_mps2.stub1**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M55.fvp_mps2.stub_i2c1**

RAM device, can be dynamic or static ram.
Type: RAMDevice.

**FVP_MPS2_Cortex_M55.fvp_mps2.stub_i2s**

RAM device, can be dynamic or static ram.
Type: RAMDevice.

**FVP_MPS2_Cortex_M55.fvp_mps2.stub_spi0**

RAM device, can be dynamic or static ram.
Type: RAMDevice.

**FVP_MPS2_Cortex_M55.fvp_mps2.stub_spi2**

RAM device, can be dynamic or static ram.
Type: RAMDevice.

**FVP_MPS2_Cortex_M55.fvp_mps2.svos_dualtimer**

Type: SVOS_DualTimer.

**FVP_MPS2_Cortex_M55.fvp_mps2.svos_dualtimer.svos_dualtimer0**

ARM Dual-Timer Module.
Type: CMSDK_DualTimer.

**FVP_MPS2_Cortex_M55.fvp_mps2.svos_dualtimer.svos_dualtimer0.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: ClockDivider.

**FVP_MPS2_Cortex_M55.fvp_mps2.svos_dualtimer.svos_dualtimer0.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: ClockDivider.

**FVP_MPS2_Cortex_M55.fvp_mps2.svos_dualtimer.svos_dualtimer0.counter0**

Internal component used by SP804 Timer module.
Type: CounterModule.

**FVP_MPS2_Cortex_M55.fvp_mps2.svos_dualtimer.svos_dualtimer0.counter1**

Internal component used by SP804 Timer module.
Type: CounterModule.

**FVP_MPS2_Cortex_M55.fvp_mps2.svos_dualtimer.svos_dualtimer1**

ARM Dual-Timer Module.
Type: CMSDK_DualTimer.

**FVP_MPS2_Cortex_M55.fvp_mps2.svos_dualtimer.svos_dualtimer1.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M55.fvp_mps2.svos_dualtimer.svos_dualtimer1.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M55.fvp_mps2.svos_dualtimer.svos_dualtimer1.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M55.fvp_mps2.svos_dualtimer.svos_dualtimer1.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M55.fvp_mps2.svos_dualtimer.svos_dualtimer2**

ARM Dual-Timer Module.
Type: `CMSDK_DualTimer`.

**FVP_MPS2_Cortex_M55.fvp_mps2.svos_dualtimer.svos_dualtimer2.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M55.fvp_mps2.svos_dualtimer.svos_dualtimer2.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M55.fvp_mps2.svos_dualtimer.svos_dualtimer2.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M55.fvp_mps2.svos_dualtimer.svos_dualtimer2.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M55.fvp_mps2.svos_dualtimer.svos_dualtimer3**

ARM Dual-Timer Module.
Type: `CMSDK_DualTimer`.

**FVP_MPS2_Cortex_M55.fvp_mps2.svos_dualtimer.svos_dualtimer3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M55.fvp_mps2.svos_dualtimer.svos_dualtimer3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M55.fvp_mps2.svos_dualtimer.svos_dualtimer3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M55.fvp_mps2.svos_dualtimer.svos_dualtimer3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M55.fvp_mps2.switch_PSRAM_M7**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M55.fvp_mps2.switch_svos_dualtimer**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M55.fvp_mps2.telnetterminal0**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_MPS2_Cortex_M55.fvp_mps2.telnetterminal1**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_MPS2_Cortex_M55.fvp_mps2.telnetterminal2**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_MPS2_Cortex_M55.fvp_mps2.touchscreen_interface**

MPS2 Touch Screen.
Type: `MPS2_TouchScreen`.

**FVP_MPS2_Cortex_M55.fvp_mps2.uart_overflows_or_gate**

Or Gate.
Type: `OrGate`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

# 16.10 FVP_MPS2_Cortex-M7

List of instances in FVP_MPS2_Cortex-M7.

## FVP_MPS2_Cortex-M7 instances

**FVP_MPS2_Cortex_M7**

Type: `FVP_MPS2_Cortex_M7`.

**FVP_MPS2_Cortex_M7.armcortexm7ct**

ARM CORTEXM7 CT model.
Type: `ARM_Cortex-M7`.

**FVP_MPS2_Cortex_M7.clk25Mhz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M7.clk25khz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M7.fvp_mps2**

MPS2 DUT.
Type: `FVP_MPS2`.

**FVP_MPS2_Cortex_M7.fvp_mps2.GPIO0**

ARM PrimeCell General Purpose Input/Output.
Type: `CMSDK_GPIO`.

**FVP_MPS2_Cortex_M7.fvp_mps2.GPIO1**

ARM PrimeCell General Purpose Input/Output.
Type: `CMSDK_GPIO`.

**FVP_MPS2_Cortex_M7.fvp_mps2.GPIO2**

ARM PrimeCell General Purpose Input/Output.
Type: `CMSDK_GPIO`.

**FVP_MPS2_Cortex_M7.fvp_mps2.GPIO3**

ARM PrimeCell General Purpose Input/Output.
Type: `CMSDK_GPIO`.

**FVP_MPS2_Cortex_M7.fvp_mps2.GPIO_connection_test**

Type: `GPIO_Connection_Test`.

**FVP_MPS2_Cortex_M7.fvp_mps2.GPIO_connection_test.GPIO0_port_trans**

Type: `GPIO_Port_Transfer`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M7.fvp_mps2.GPIO_connection_test.GPIO1_port_test**

Type: `GPIO1_Connection_Test`.

**FVP_MPS2_Cortex_M7.fvp_mps2.PSRAM**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M7.fvp_mps2.PSRAM_M7**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M7.fvp_mps2.UART0**

ARM CMSDK UART Module.
Type: `CMSDK_UART`.

**FVP_MPS2_Cortex_M7.fvp_mps2.UART0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M7.fvp_mps2.UART1**

ARM CMSDK UART Module.
Type: `CMSDK_UART`.

**FVP_MPS2_Cortex_M7.fvp_mps2.UART1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M7.fvp_mps2.UART2**

ARM CMSDK UART Module.
Type: `CMSDK_UART`.

**FVP_MPS2_Cortex_M7.fvp_mps2.UART2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M7.fvp_mps2.VGA_interface**

VGA display interface between main bus and visualisation.
Type: `MPS2_VGA`.

**FVP_MPS2_Cortex_M7.fvp_mps2.ahb_ppc_iotss_expansion0**

IoT Subsystem Peripheral Protection Controller.
Type: `IoTSS_PeripheralProtectionController`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M7.fvp_mps2.ahb_ppc_iotss_expansion1**
IoT Subsystem Peripheral Protection Controller.
Type: `IoTSS_PeripheralProtectionController`.

**FVP_MPS2_Cortex_M7.fvp_mps2.apb_ppc_iotss_expansion0**
IoT Subsystem Peripheral Protection Controller.
Type: `IoTSS_PeripheralProtectionController`.

**FVP_MPS2_Cortex_M7.fvp_mps2.apb_ppc_iotss_expansion1**
IoT Subsystem Peripheral Protection Controller.
Type: `IoTSS_PeripheralProtectionController`.

**FVP_MPS2_Cortex_M7.fvp_mps2.apb_ppc_iotss_expansion2**
IoT Subsystem Peripheral Protection Controller.
Type: `IoTSS_PeripheralProtectionController`.

**FVP_MPS2_Cortex_M7.fvp_mps2.clock50Hz**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M7.fvp_mps2.clockdivider**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M7.fvp_mps2.cmsdk_sysctrl**
Cortex-M Simple System Control.
Type: `CMSDK_SysCtrl`.

**FVP_MPS2_Cortex_M7.fvp_mps2.cmsdk_watchdog**
ARM Watchdog Module.
Type: `CMSDK_Watchdog`.

**FVP_MPS2_Cortex_M7.fvp_mps2.cpu_wait_or_gate_0**
Or Gate.
Type: `OrGate`.

**FVP_MPS2_Cortex_M7.fvp_mps2.cpu_wait_or_gate_1**
Or Gate.
Type: `OrGate`.

**FVP_MPS2_Cortex_M7.fvp_mps2.dbgen_or_gate**
Or Gate.
Type: `OrGate`.

**FVP_MPS2_Cortex_M7.fvp_mps2.dma0**
ARM PrimeCell DMA Controller(PL080/081).
Type: `PL080_DMAC`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M7.fvp_mps2.dma0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_MPS2_Cortex_M7.fvp_mps2.dma0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_MPS2_Cortex_M7.fvp_mps2.dma0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_MPS2_Cortex_M7.fvp_mps2.dma0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M7.fvp_mps2.dma0_idau_labeller**

Type: `LabellerIdauSecurity`.

**FVP_MPS2_Cortex_M7.fvp_mps2.dma0_securitymodifier**

Type: `SecurityModifier`.

**FVP_MPS2_Cortex_M7.fvp_mps2.dma1**

ARM PrimeCell DMA Controller(PL080/081).
Type: `PL080_DMAC`.

**FVP_MPS2_Cortex_M7.fvp_mps2.dma1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_MPS2_Cortex_M7.fvp_mps2.dma1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_MPS2_Cortex_M7.fvp_mps2.dma1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_MPS2_Cortex_M7.fvp_mps2.dma1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M7.fvp_mps2.dma1_idau_labeller**

Type: `LabellerIdauSecurity`.

**FVP_MPS2_Cortex_M7.fvp_mps2.dma1_securitymodifier**

Type: `SecurityModifier`.

**FVP_MPS2_Cortex_M7.fvp_mps2.dma2**

ARM PrimeCell DMA Controller(PL080/081).
Type: `PL080_DMAC`.

**FVP_MPS2_Cortex_M7.fvp_mps2.dma2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_MPS2_Cortex_M7.fvp_mps2.dma2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_MPS2_Cortex_M7.fvp_mps2.dma2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_MPS2_Cortex_M7.fvp_mps2.dma2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M7.fvp_mps2.dma2_idau_labeller**

Type: `LabellerIdauSecurity`.

**FVP_MPS2_Cortex_M7.fvp_mps2.dma2_securitymodifier**

Type: `SecurityModifier`.

**FVP_MPS2_Cortex_M7.fvp_mps2.dma3**

ARM PrimeCell DMA Controller(PL080/081).
Type: `PL080_DMAC`.

**FVP_MPS2_Cortex_M7.fvp_mps2.dma3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_MPS2_Cortex_M7.fvp_mps2.dma3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_MPS2_Cortex_M7.fvp_mps2.dma3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_MPS2_Cortex_M7.fvp_mps2.dma3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M7.fvp_mps2.dma3_idau_labeller**

Type: `LabellerIdauSecurity`.

**FVP_MPS2_Cortex_M7.fvp_mps2.dma3_securitymodifier**

Type: `SecurityModifier`.

**FVP_MPS2_Cortex_M7.fvp_mps2.exclusive_monitor_psram**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_Cortex_M7.fvp_mps2.exclusive_monitor_psram_iotss**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M7.fvp_mps2.exclusive_monitor_switch_extra_psram_iotss**
Allow transactions to be routed arbitrarily.
Type: PVBusRouter.

**FVP_MPS2_Cortex_M7.fvp_mps2.exclusive_monitor_switch_extra_psram_mps2**
Allow transactions to be routed arbitrarily.
Type: PVBusRouter.

**FVP_MPS2_Cortex_M7.fvp_mps2.exclusive_monitor_zbtsram1**
Global exclusive monitor.
Type: PVBusExclusiveMonitor.

**FVP_MPS2_Cortex_M7.fvp_mps2.exclusive_monitor_zbtsram2**
Global exclusive monitor.
Type: PVBusExclusiveMonitor.

**FVP_MPS2_Cortex_M7.fvp_mps2.exclusive_squasher**
Squashes the exclusive attribute on bus transactions.
Type: PVBusExclusiveSquasher.

**FVP_MPS2_Cortex_M7.fvp_mps2.fpga_sysctrl**
FPGA SysCtrl timers LEDs and switches.
Type: FPGA_SysCtrl.

**FVP_MPS2_Cortex_M7.fvp_mps2.fpga_sysctrl.callBack100HzCounter**
Internal component used by SP804 Timer module.
Type: CounterModule.

**FVP_MPS2_Cortex_M7.fvp_mps2.fpga_sysctrl.clockdivider**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: ClockDivider.

**FVP_MPS2_Cortex_M7.fvp_mps2.gpio_0_or_2**
Or Gate.
Type: OrGate.

**FVP_MPS2_Cortex_M7.fvp_mps2.gpio_1_or_3**
Or Gate.
Type: OrGate.

**FVP_MPS2_Cortex_M7.fvp_mps2.hostbridge**
Host Socket Interface Component.
Type: HostBridge.

**FVP_MPS2_Cortex_M7.fvp_mps2.mem_switch_extra_psram_iotss**
Allow transactions to be routed arbitrarily.
Type: PVBusRouter.

**FVP_MPS2_Cortex_M7.fvp_mps2.mem_switch_extra_psram_mps2**
Allow transactions to be routed arbitrarily.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M7.fvp_mps2.mpc_iotss_ssram1**

IoT Subsystem Memory Protection Controller.
Type: `IoTSS_MemoryProtectionController`.

**FVP_MPS2_Cortex_M7.fvp_mps2.mpc_iotss_ssram1.gating_disabled_thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M7.fvp_mps2.mpc_iotss_ssram2**

IoT Subsystem Memory Protection Controller.
Type: `IoTSS_MemoryProtectionController`.

**FVP_MPS2_Cortex_M7.fvp_mps2.mpc_iotss_ssram2.gating_disabled_thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M7.fvp_mps2.mpc_iotss_ssram3**

IoT Subsystem Memory Protection Controller.
Type: `IoTSS_MemoryProtectionController`.

**FVP_MPS2_Cortex_M7.fvp_mps2.mpc_iotss_ssram3.gating_disabled_thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M7.fvp_mps2.mps2_audio**

MPS2 Audio.
Type: `MPS2_Audio`.

**FVP_MPS2_Cortex_M7.fvp_mps2.mps2_cmsdk_dualtimer**

ARM Dual-Timer Module.
Type: `CMSDK_DualTimer`.

**FVP_MPS2_Cortex_M7.fvp_mps2.mps2_cmsdk_dualtimer.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M7.fvp_mps2.mps2_cmsdk_dualtimer.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M7.fvp_mps2.mps2_cmsdk_dualtimer.counter0**

Internal component used by SP804 Timer module.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `CounterModule`.

**FVP_MPS2_Cortex_M7.fvp_mps2.mps2_cmsdk_dualtimer.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M7.fvp_mps2.mps2_exclusive_monitor_zbtsram1**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_Cortex_M7.fvp_mps2.mps2_exclusive_monitor_zbtsram2**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_Cortex_M7.fvp_mps2.mps2_lcd**

MPS2 LCD I2C interface.
Type: `MPS2_LCD`.

**FVP_MPS2_Cortex_M7.fvp_mps2.mps2_mpc_iotss_ssram1**

IoT Subsystem Memory Protection Controller.
Type: `IoTSS_MemoryProtectionController`.

**FVP_MPS2_Cortex_M7.fvp_mps2.mps2_mpc_iotss_ssram1.gating_disabled_thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M7.fvp_mps2.mps2_secure_control_register_block**

MPS2 Secure Control Register Block.
Type: `MPS2_SecureCtrl`.

**FVP_MPS2_Cortex_M7.fvp_mps2.mps2_timer0**

ARM Timer Module.
Type: `CMSDK_Timer`.

**FVP_MPS2_Cortex_M7.fvp_mps2.mps2_timer0.clk_div**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M7.fvp_mps2.mps2_timer0.counter**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M7.fvp_mps2.mps2_timer1**

ARM Timer Module.
Type: `CMSDK_Timer`.

**FVP_MPS2_Cortex_M7.fvp_mps2.mps2_timer1.clk_div**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M7.fvp_mps2.mps2_timer1.counter**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M7.fvp_mps2.mps2_visualisation**

Display window for MPS2 using Visualisation library.
Type: `MPS2_Visualisation`.

**FVP_MPS2_Cortex_M7.fvp_mps2.niden_or_gate**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_Cortex_M7.fvp_mps2.nmi_or_gate**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_Cortex_M7.fvp_mps2.pl022_ssp_mps2**

ARM PrimeCell Synchronous Serial Port(PL022).
Type: `PL022_SSP_MPS2`.

**FVP_MPS2_Cortex_M7.fvp_mps2.pl022_ssp_mps2.prescaler**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M7.fvp_mps2.platform_bus_switch**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M7.fvp_mps2.platform_switch_dma0**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M7.fvp_mps2.platform_switch_dma1**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M7.fvp_mps2.signal_router**

Signal router.
Type: `SignalRouter`.

**FVP_MPS2_Cortex_M7.fvp_mps2.smsc_91c111**

SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_MPS2_Cortex_M7.fvp_mps2.spiden_or_gate**

Or Gate.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `OrGate`.

**FVP_MPS2_Cortex_M7.fvp_mps2.spniden_or_gate**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_Cortex_M7.fvp_mps2.sse200**

SSE-200 subsystem.
Type: `SSE200`.

**FVP_MPS2_Cortex_M7.fvp_mps2.sse200.acg_cpu0**

IoT Subsystem Access Control Gate.
Type: `IoTSS_AccessControlGate`.

**FVP_MPS2_Cortex_M7.fvp_mps2.sse200.acg_cpu1**

IoT Subsystem Access Control Gate.
Type: `IoTSS_AccessControlGate`.

**FVP_MPS2_Cortex_M7.fvp_mps2.sse200.acg_sram0**

IoT Subsystem Access Control Gate.
Type: `IoTSS_AccessControlGate`.

**FVP_MPS2_Cortex_M7.fvp_mps2.sse200.acg_sram1**

IoT Subsystem Access Control Gate.
Type: `IoTSS_AccessControlGate`.

**FVP_MPS2_Cortex_M7.fvp_mps2.sse200.acg_sram2**

IoT Subsystem Access Control Gate.
Type: `IoTSS_AccessControlGate`.

**FVP_MPS2_Cortex_M7.fvp_mps2.sse200.acg_sram3**

IoT Subsystem Access Control Gate.
Type: `IoTSS_AccessControlGate`.

**FVP_MPS2_Cortex_M7.fvp_mps2.sse200.apb_ppc_iotss_subsystem0**

IoT Subsystem Peripheral Protection Controller.
Type: `IoTSS_PeripheralProtectionController`.

**FVP_MPS2_Cortex_M7.fvp_mps2.sse200.apb_ppc_iotss_subsystem1**

IoT Subsystem Peripheral Protection Controller.
Type: `IoTSS_PeripheralProtectionController`.

**FVP_MPS2_Cortex_M7.fvp_mps2.sse200.clock32kHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M7.fvp_mps2.sse200.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M7.fvp_mps2.sse200.cmsdk_dualtimer**

ARM Dual-Timer Module.
Type: `CMSDK_DualTimer`.

**FVP_MPS2_Cortex_M7.fvp_mps2.sse200.cmsdk_dualtimer.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M7.fvp_mps2.sse200.cmsdk_dualtimer.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M7.fvp_mps2.sse200.cmsdk_dualtimer.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M7.fvp_mps2.sse200.cmsdk_dualtimer.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M7.fvp_mps2.sse200.cordio_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M7.fvp_mps2.sse200.cpu0core_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M7.fvp_mps2.sse200.cpu0dbg_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M7.fvp_mps2.sse200.cpu1core_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M7.fvp_mps2.sse200.cpu1dbg_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M7.fvp_mps2.sse200.crypto_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M7.fvp_mps2.sse200.dbg_ppu**
ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M7.fvp_mps2.sse200.exclusive_monitor_iotss_internal_sram0**
Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_Cortex_M7.fvp_mps2.sse200.exclusive_monitor_iotss_internal_sram1**
Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_Cortex_M7.fvp_mps2.sse200.exclusive_monitor_iotss_internal_sram2**
Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_Cortex_M7.fvp_mps2.sse200.exclusive_monitor_iotss_internal_sram3**
Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_Cortex_M7.fvp_mps2.sse200.exclusive_squasher**
Squashes the exclusive attribute on bus transactions.
Type: `PVBusExclusiveSquasher`.

**FVP_MPS2_Cortex_M7.fvp_mps2.sse200.idau_labeller**
Type: `LabellerIdauSecurity`.

**FVP_MPS2_Cortex_M7.fvp_mps2.sse200.internal_msniper_tcm**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M7.fvp_mps2.sse200.iotss_cpuidentity**
IoT Subsystem CPU_IDENTITY registers.
Type: `IoTSS_CPUIdentity`.

**FVP_MPS2_Cortex_M7.fvp_mps2.sse200.iotss_internal_sram0**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M7.fvp_mps2.sse200.iotss_internal_sram1**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M7.fvp_mps2.sse200.iotss_internal_sram2**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M7.fvp_mps2.sse200.iotss_internal_sram3**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M7.fvp_mps2.sse200.iotss_systemcontrol**
IoT Subsystem System Control registers.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `IoTSS_SystemControl`.

**FVP_MPS2_Cortex_M7.fvp_mps2.sse200.iotss_systeminfo**

IoT Subsystem System Information registers.
Type: `IoTSS_SystemInfo`.

**FVP_MPS2_Cortex_M7.fvp_mps2.sse200.mem_switch_internal_msniper_tcm**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M7.fvp_mps2.sse200.mem_switch_internal_sram**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M7.fvp_mps2.sse200.mem_switch_internal_sram_mpc**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M7.fvp_mps2.sse200.mem_switch_ppu**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M7.fvp_mps2.sse200.mhu0**

IoT Subsystem Message Handling Unit.
Type: `IoTSS_MessageHandlingUnit`.

**FVP_MPS2_Cortex_M7.fvp_mps2.sse200.mhu1**

IoT Subsystem Message Handling Unit.
Type: `IoTSS_MessageHandlingUnit`.

**FVP_MPS2_Cortex_M7.fvp_mps2.sse200.mpc_iotss_internal_sram0**

IoT Subsystem Memory Protection Controller.
Type: `IoTSS_MemoryProtectionController`.

**FVP_MPS2_Cortex_M7.fvp_mps2.sse200.mpc_iotss_internal_sram0.gating_disabled_thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M7.fvp_mps2.sse200.mpc_iotss_internal_sram1**

IoT Subsystem Memory Protection Controller.
Type: `IoTSS_MemoryProtectionController`.

**FVP_MPS2_Cortex_M7.fvp_mps2.sse200.mpc_iotss_internal_sram1.gating_disabled_thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M7.fvp_mps2.sse200.mpc_iotss_internal_sram2**

IoT Subsystem Memory Protection Controller.
Type: `IoTSS_MemoryProtectionController`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M7.fvp_mps2.sse200.mpc_iotss_internal_sram2.gating_disabled_thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M7.fvp_mps2.sse200.mpc_iotss_internal_sram3**

IoT Subsystem Memory Protection Controller.
Type: `IoTSS_MemoryProtectionController`.

**FVP_MPS2_Cortex_M7.fvp_mps2.sse200.mpc_iotss_internal_sram3.gating_disabled_thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M7.fvp_mps2.sse200.nmi_or_gate**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_Cortex_M7.fvp_mps2.sse200.nonsecure_watchdog**

ARM Watchdog Module.
Type: `CMSDK_Watchdog`.

**FVP_MPS2_Cortex_M7.fvp_mps2.sse200.ram0_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M7.fvp_mps2.sse200.ram1_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M7.fvp_mps2.sse200.ram2_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M7.fvp_mps2.sse200.ram3_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M7.fvp_mps2.sse200.s32k_timer**

ARM Timer Module.
Type: `CMSDK_Timer`.

**FVP_MPS2_Cortex_M7.fvp_mps2.sse200.s32k_timer.clk_div**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M7.fvp_mps2.sse200.s32k_timer.counter**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M7.fvp_mps2.sse200.s32k_watchdog**
ARM Watchdog Module.
Type: `CMSDK_Watchdog`.

**FVP_MPS2_Cortex_M7.fvp_mps2.sse200.secure_control_register_block**
MPS2 Secure Control Register Block.
Type: `MPS2_SecureCtrl`.

**FVP_MPS2_Cortex_M7.fvp_mps2.sse200.secure_watchdog**
ARM Watchdog Module.
Type: `CMSDK_Watchdog`.

**FVP_MPS2_Cortex_M7.fvp_mps2.sse200.signal_router**
Signal router.
Type: `SignalRouter`.

**FVP_MPS2_Cortex_M7.fvp_mps2.sse200.sys_ppu**
ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M7.fvp_mps2.sse200.timer0**
ARM Timer Module.
Type: `CMSDK_Timer`.

**FVP_MPS2_Cortex_M7.fvp_mps2.sse200.timer0.clk_div**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M7.fvp_mps2.sse200.timer0.counter**
Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M7.fvp_mps2.sse200.timer1**
ARM Timer Module.
Type: `CMSDK_Timer`.

**FVP_MPS2_Cortex_M7.fvp_mps2.sse200.timer1.clk_div**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M7.fvp_mps2.sse200.timer1.counter**
Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M7.fvp_mps2.ssram1**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M7.fvp_mps2.ssram2**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M7.fvp_mps2.stub0**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M7.fvp_mps2.stub1**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M7.fvp_mps2.stub_i2c1**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M7.fvp_mps2.stub_i2s**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M7.fvp_mps2.stub_spi0**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M7.fvp_mps2.stub_spi2**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M7.fvp_mps2.svos_dualtimer**
Type: `SVOS_DualTimer`.

**FVP_MPS2_Cortex_M7.fvp_mps2.svos_dualtimer.svos_dualtimer0**
ARM Dual-Timer Module.
Type: `CMSDK_DualTimer`.

**FVP_MPS2_Cortex_M7.fvp_mps2.svos_dualtimer.svos_dualtimer0.clk_div0**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M7.fvp_mps2.svos_dualtimer.svos_dualtimer0.clk_div1**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M7.fvp_mps2.svos_dualtimer.svos_dualtimer0.counter0**
Internal component used by SP804 Timer module.
Type: `CounterModule`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M7.fvp_mps2.svos_dualtimer.svos_dualtimer0.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M7.fvp_mps2.svos_dualtimer.svos_dualtimer1**

ARM Dual-Timer Module.
Type: `CMSDK_DualTimer`.

**FVP_MPS2_Cortex_M7.fvp_mps2.svos_dualtimer.svos_dualtimer1.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M7.fvp_mps2.svos_dualtimer.svos_dualtimer1.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M7.fvp_mps2.svos_dualtimer.svos_dualtimer1.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M7.fvp_mps2.svos_dualtimer.svos_dualtimer1.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M7.fvp_mps2.svos_dualtimer.svos_dualtimer2**

ARM Dual-Timer Module.
Type: `CMSDK_DualTimer`.

**FVP_MPS2_Cortex_M7.fvp_mps2.svos_dualtimer.svos_dualtimer2.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M7.fvp_mps2.svos_dualtimer.svos_dualtimer2.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M7.fvp_mps2.svos_dualtimer.svos_dualtimer2.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M7.fvp_mps2.svos_dualtimer.svos_dualtimer2.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M7.fvp_mps2.svos_dualtimer.svos_dualtimer3**

ARM Dual-Timer Module.
Type: `CMSDK_DualTimer`.

**FVP_MPS2_Cortex_M7.fvp_mps2.svos_dualtimer.svos_dualtimer3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M7.fvp_mps2.svos_dualtimer.svos_dualtimer3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M7.fvp_mps2.svos_dualtimer.svos_dualtimer3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M7.fvp_mps2.svos_dualtimer.svos_dualtimer3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M7.fvp_mps2.switch_PSRAM_M7**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M7.fvp_mps2.switch_svos_dualtimer**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M7.fvp_mps2.telnetterminal0**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_MPS2_Cortex_M7.fvp_mps2.telnetterminal1**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_MPS2_Cortex_M7.fvp_mps2.telnetterminal2**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_MPS2_Cortex_M7.fvp_mps2.touchscreen_interface**

MPS2 Touch Screen.
Type: `MPS2_TouchScreen`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M7.fvp_mps2.uart_overflows_or_gate**

Or Gate.
Type: `OrGate`.

# 16.11 FVP_MPS2_Cortex-M85

List of instances in FVP_MPS2_Cortex-M85.

## Differences between the FVP and the MPS2 specification

- The model does not have the random number generator or unique ID/secure storage mentioned in the MPS2 specification because the programmer's view of these devices is yet to be defined.
- The Ethernet controller in the model is a LAN91C111. The MPS2 documents, including those for pre-v8-M cores, specify a LAN9220.
- MTB, ETM, and TPIU are not supported. MTB RAM is absent.
- In the Memory Gating Unit, the model provides a configurable block size. For performance reasons, the minimum block size in the model is 4096 bytes. Hardware and later models might allow smaller block sizes. Software should always use the `BLK_CFG` register to determine block size.
- As in other MPS2 Fast Models, some of the peripherals have minimal implementations:
  - The Audio controller is RAZ/WI.
  - Only the touchscreen functionality of the STMPE811 touchscreen controller is implemented.
  - A subset of the Ampire LCD module's graphics modes are supported.
- MPS2 Phase 2 includes features to enable the creation of complex software. However, for those new to TrustZone® for Arm® Cortex®-M, it might be useful to run the model with some of these features disabled. For example, the following parameters disable the IDAU and the security gates:

```
FVP_MPS2_Cortex-M85 --parameter idau.NUM_IDAU_REGION=0 --parameter
 fvp_mps2.DISABLE_GATING=1
```

- See the full set of model parameters and their descriptions by running the model with the `--list-params` argument. Most parameters share their names with a corresponding RTL configuration parameter.

## FVP_MPS2_Cortex-M85 instances

**FVP_MPS2_Cortex_M85**

Type: `FVP_MPS2_Cortex_M85`.

**FVP_MPS2_Cortex_M85.clk25Mhz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M85.clk25khz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M85.core0_bus_gasket**

writes are ignored, non-word reads abort, and word reads take values from that fifo.
Type: `STLBusGasket`.

**FVP_MPS2_Cortex_M85.cpu0**

ARM Cortex-M85 CT model.
Type: `ARM_Cortex-M85`.

**FVP_MPS2_Cortex_M85.cpu1**

ARM Cortex-M85 CT model.
Type: `ARM_Cortex-M85`.

**FVP_MPS2_Cortex_M85.fvp_mps2**

MPS2 DUT.
Type: `FVP_MPS2`.

**FVP_MPS2_Cortex_M85.fvp_mps2.GPIO0**

ARM PrimeCell General Purpose Input/Output.
Type: `CMSDK_GPIO`.

**FVP_MPS2_Cortex_M85.fvp_mps2.GPIO1**

ARM PrimeCell General Purpose Input/Output.
Type: `CMSDK_GPIO`.

**FVP_MPS2_Cortex_M85.fvp_mps2.GPIO2**

ARM PrimeCell General Purpose Input/Output.
Type: `CMSDK_GPIO`.

**FVP_MPS2_Cortex_M85.fvp_mps2.GPIO3**

ARM PrimeCell General Purpose Input/Output.
Type: `CMSDK_GPIO`.

**FVP_MPS2_Cortex_M85.fvp_mps2.GPIO_connection_test**

Type: `GPIO_Connection_Test`.

**FVP_MPS2_Cortex_M85.fvp_mps2.GPIO_connection_test.GPIO0_port_trans**

Type: `GPIO_Port_Transfer`.

**FVP_MPS2_Cortex_M85.fvp_mps2.GPIO_connection_test.GPIO1_port_test**

Type: `GPIO1_Connection_Test`.

**FVP_MPS2_Cortex_M85.fvp_mps2.PSRAM**

RAM device, can be dynamic or static ram.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `RAMDevice`.

**FVP_MPS2_Cortex_M85.fvp_mps2.PSRAM_M7**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M85.fvp_mps2.UART0**

ARM CMSDK UART Module.
Type: `CMSDK_UART`.

**FVP_MPS2_Cortex_M85.fvp_mps2.UART0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M85.fvp_mps2.UART1**

ARM CMSDK UART Module.
Type: `CMSDK_UART`.

**FVP_MPS2_Cortex_M85.fvp_mps2.UART1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M85.fvp_mps2.UART2**

ARM CMSDK UART Module.
Type: `CMSDK_UART`.

**FVP_MPS2_Cortex_M85.fvp_mps2.UART2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M85.fvp_mps2.VGA_interface**

VGA display interface between main bus and visualisation.
Type: `MPS2_VGA`.

**FVP_MPS2_Cortex_M85.fvp_mps2.ahb_ppc_iotss_expansion0**

IoT Subsystem Peripheral Protection Controller.
Type: `IoTSS_PeripheralProtectionController`.

**FVP_MPS2_Cortex_M85.fvp_mps2.ahb_ppc_iotss_expansion1**

IoT Subsystem Peripheral Protection Controller.
Type: `IoTSS_PeripheralProtectionController`.

**FVP_MPS2_Cortex_M85.fvp_mps2.apb_ppc_iotss_expansion0**

IoT Subsystem Peripheral Protection Controller.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `IoTSS_PeripheralProtectionController`.

**FVP_MPS2_Cortex_M85.fvp_mps2.apb_ppc_iotss_expansion1**

IoT Subsystem Peripheral Protection Controller.
Type: `IoTSS_PeripheralProtectionController`.

**FVP_MPS2_Cortex_M85.fvp_mps2.apb_ppc_iotss_expansion2**

IoT Subsystem Peripheral Protection Controller.
Type: `IoTSS_PeripheralProtectionController`.

**FVP_MPS2_Cortex_M85.fvp_mps2.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M85.fvp_mps2.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M85.fvp_mps2.cmsdk_sysctrl**

Cortex-M Simple System Control.
Type: `CMSDK_SysCtrl`.

**FVP_MPS2_Cortex_M85.fvp_mps2.cmsdk_watchdog**

ARM Watchdog Module.
Type: `CMSDK_Watchdog`.

**FVP_MPS2_Cortex_M85.fvp_mps2.cpu_wait_or_gate_0**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_Cortex_M85.fvp_mps2.cpu_wait_or_gate_1**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_Cortex_M85.fvp_mps2.dbgen_or_gate**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_Cortex_M85.fvp_mps2.dma0**

ARM PrimeCell DMA Controller(PL080/081).
Type: `PL080_DMAC`.

**FVP_MPS2_Cortex_M85.fvp_mps2.dma0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_MPS2_Cortex_M85.fvp_mps2.dma0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_MPS2_Cortex_M85.fvp_mps2.dma0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_MPS2_Cortex_M85.fvp_mps2.dma0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M85.fvp_mps2.dma0_idau_labeller**

Type: `LabellerIdauSecurity`.

**FVP_MPS2_Cortex_M85.fvp_mps2.dma0_securitymodifier**

Type: `SecurityModifier`.

**FVP_MPS2_Cortex_M85.fvp_mps2.dma1**

ARM PrimeCell DMA Controller(PL080/081).
Type: `PL080_DMAC`.

**FVP_MPS2_Cortex_M85.fvp_mps2.dma1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_MPS2_Cortex_M85.fvp_mps2.dma1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M85.fvp_mps2.dma1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_MPS2_Cortex_M85.fvp_mps2.dma1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M85.fvp_mps2.dma1_idau_labeller**

Type: `LabellerIdauSecurity`.

**FVP_MPS2_Cortex_M85.fvp_mps2.dma1_securitymodifier**

Type: `SecurityModifier`.

**FVP_MPS2_Cortex_M85.fvp_mps2.dma2**

ARM PrimeCell DMA Controller(PL080/081).
Type: `PL080_DMAC`.

**FVP_MPS2_Cortex_M85.fvp_mps2.dma2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_MPS2_Cortex_M85.fvp_mps2.dma2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_MPS2_Cortex_M85.fvp_mps2.dma2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_MPS2_Cortex_M85.fvp_mps2.dma2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M85.fvp_mps2.dma2_idau_labeller**

Type: `LabellerIdauSecurity`.

**FVP_MPS2_Cortex_M85.fvp_mps2.dma2_securitymodifier**

Type: `SecurityModifier`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M85.fvp_mps2.dma3**

ARM PrimeCell DMA Controller(PL080/081).
Type: `PL080_DMAC`.

**FVP_MPS2_Cortex_M85.fvp_mps2.dma3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_MPS2_Cortex_M85.fvp_mps2.dma3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_MPS2_Cortex_M85.fvp_mps2.dma3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_MPS2_Cortex_M85.fvp_mps2.dma3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M85.fvp_mps2.dma3_idau_labeller**

Type: `LabellerIdauSecurity`.

**FVP_MPS2_Cortex_M85.fvp_mps2.dma3_securitymodifier**

Type: `SecurityModifier`.

**FVP_MPS2_Cortex_M85.fvp_mps2.exclusive_monitor_psram**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_Cortex_M85.fvp_mps2.exclusive_monitor_psram_iotss**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_Cortex_M85.fvp_mps2.exclusive_monitor_switch_extra_psram_iotss**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M85.fvp_mps2.exclusive_monitor_switch_extra_psram_mps2**

Allow transactions to be routed arbitrarily.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M85.fvp_mps2.exclusive_monitor_zbtsram1**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_Cortex_M85.fvp_mps2.exclusive_monitor_zbtsram2**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_Cortex_M85.fvp_mps2.exclusive_squasher**

Squashes the exclusive attribute on bus transactions.
Type: `PVBusExclusiveSquasher`.

**FVP_MPS2_Cortex_M85.fvp_mps2.fpga_sysctrl**

FPGA SysCtrl timers LEDs and switches.
Type: `FPGA_SysCtrl`.

**FVP_MPS2_Cortex_M85.fvp_mps2.fpga_sysctrl.callBack100HzCounter**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M85.fvp_mps2.fpga_sysctrl.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M85.fvp_mps2.gpio_0_or_2**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_Cortex_M85.fvp_mps2.gpio_1_or_3**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_Cortex_M85.fvp_mps2.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_MPS2_Cortex_M85.fvp_mps2.mem_switch_extra_psram_iotss**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M85.fvp_mps2.mem_switch_extra_psram_mps2**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M85.fvp_mps2.mpc_iotss_ssram1**

IoT Subsystem Memory Protection Controller.
Type: `IoTSS_MemoryProtectionController`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M85.fvp_mps2.mpc_iotss_ssram1.gating_disabled_thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M85.fvp_mps2.mpc_iotss_ssram2**

IoT Subsystem Memory Protection Controller.
Type: `IoTSS_MemoryProtectionController`.

**FVP_MPS2_Cortex_M85.fvp_mps2.mpc_iotss_ssram2.gating_disabled_thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M85.fvp_mps2.mpc_iotss_ssram3**

IoT Subsystem Memory Protection Controller.
Type: `IoTSS_MemoryProtectionController`.

**FVP_MPS2_Cortex_M85.fvp_mps2.mpc_iotss_ssram3.gating_disabled_thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M85.fvp_mps2.mps2_audio**

MPS2 Audio.
Type: `MPS2_Audio`.

**FVP_MPS2_Cortex_M85.fvp_mps2.mps2_cmsdk_dualtimer**

ARM Dual-Timer Module.
Type: `CMSDK_DualTimer`.

**FVP_MPS2_Cortex_M85.fvp_mps2.mps2_cmsdk_dualtimer.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M85.fvp_mps2.mps2_cmsdk_dualtimer.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M85.fvp_mps2.mps2_cmsdk_dualtimer.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M85.fvp_mps2.mps2_cmsdk_dualtimer.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M85.fvp_mps2.mps2_exclusive_monitor_zbtsram1**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_Cortex_M85.fvp_mps2.mps2_exclusive_monitor_zbtsram2**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_Cortex_M85.fvp_mps2.mps2_lcd**

MPS2 LCD I2C interface.
Type: `MPS2_LCD`.

**FVP_MPS2_Cortex_M85.fvp_mps2.mps2_mpc_iotss_ssram1**

IoT Subsystem Memory Protection Controller.
Type: `IoTSS_MemoryProtectionController`.

**FVP_MPS2_Cortex_M85.fvp_mps2.mps2_mpc_iotss_ssram1.gating_disabled_thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M85.fvp_mps2.mps2_secure_control_register_block**

MPS2 Secure Control Register Block.
Type: `MPS2_SecureCtrl`.

**FVP_MPS2_Cortex_M85.fvp_mps2.mps2_timer0**

ARM Timer Module.
Type: `CMSDK_Timer`.

**FVP_MPS2_Cortex_M85.fvp_mps2.mps2_timer0.clk_div**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M85.fvp_mps2.mps2_timer0.counter**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M85.fvp_mps2.mps2_timer1**

ARM Timer Module.
Type: `CMSDK_Timer`.

**FVP_MPS2_Cortex_M85.fvp_mps2.mps2_timer1.clk_div**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M85.fvp_mps2.mps2_timer1.counter**

Internal component used by SP804 Timer module.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `CounterModule`.

**FVP_MPS2_Cortex_M85.fvp_mps2.mps2_visualisation**

Display window for MPS2 using Visualisation library.
Type: `MPS2_Visualisation`.

**FVP_MPS2_Cortex_M85.fvp_mps2.niden_or_gate**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_Cortex_M85.fvp_mps2.nmi_or_gate**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_Cortex_M85.fvp_mps2.pl022_ssp_mps2**

ARM PrimeCell Synchronous Serial Port(PL022).
Type: `PL022_SSP_MPS2`.

**FVP_MPS2_Cortex_M85.fvp_mps2.pl022_ssp_mps2.prescaler**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M85.fvp_mps2.platform_bus_switch**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M85.fvp_mps2.platform_switch_dma0**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M85.fvp_mps2.platform_switch_dma1**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M85.fvp_mps2.signal_router**

Signal router.
Type: `SignalRouter`.

**FVP_MPS2_Cortex_M85.fvp_mps2.smsc_91c111**

SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_MPS2_Cortex_M85.fvp_mps2.spiden_or_gate**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_Cortex_M85.fvp_mps2.spniden_or_gate**

Or Gate.
Type: `OrGate`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M85.fvp_mps2.sse200**

SSE-200 subsystem.
Type: `SSE200`.

**FVP_MPS2_Cortex_M85.fvp_mps2.sse200.acg_cpu0**

IoT Subsystem Access Control Gate.
Type: `IoTSS_AccessControlGate`.

**FVP_MPS2_Cortex_M85.fvp_mps2.sse200.acg_cpu1**

IoT Subsystem Access Control Gate.
Type: `IoTSS_AccessControlGate`.

**FVP_MPS2_Cortex_M85.fvp_mps2.sse200.acg_sram0**

IoT Subsystem Access Control Gate.
Type: `IoTSS_AccessControlGate`.

**FVP_MPS2_Cortex_M85.fvp_mps2.sse200.acg_sram1**

IoT Subsystem Access Control Gate.
Type: `IoTSS_AccessControlGate`.

**FVP_MPS2_Cortex_M85.fvp_mps2.sse200.acg_sram2**

IoT Subsystem Access Control Gate.
Type: `IoTSS_AccessControlGate`.

**FVP_MPS2_Cortex_M85.fvp_mps2.sse200.acg_sram3**

IoT Subsystem Access Control Gate.
Type: `IoTSS_AccessControlGate`.

**FVP_MPS2_Cortex_M85.fvp_mps2.sse200.apb_ppc_iotss_subsystem0**

IoT Subsystem Peripheral Protection Controller.
Type: `IoTSS_PeripheralProtectionController`.

**FVP_MPS2_Cortex_M85.fvp_mps2.sse200.apb_ppc_iotss_subsystem1**

IoT Subsystem Peripheral Protection Controller.
Type: `IoTSS_PeripheralProtectionController`.

**FVP_MPS2_Cortex_M85.fvp_mps2.sse200.clock32kHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M85.fvp_mps2.sse200.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M85.fvp_mps2.sse200.cmsdk_dualtimer**

ARM Dual-Timer Module.
Type: `CMSDK_DualTimer`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M85.fvp_mps2.sse200.cmsdk_dualtimer.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M85.fvp_mps2.sse200.cmsdk_dualtimer.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M85.fvp_mps2.sse200.cmsdk_dualtimer.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M85.fvp_mps2.sse200.cmsdk_dualtimer.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M85.fvp_mps2.sse200.cordio_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M85.fvp_mps2.sse200.cpu0core_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M85.fvp_mps2.sse200.cpu0dbg_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M85.fvp_mps2.sse200.cpu1core_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M85.fvp_mps2.sse200.cpu1dbg_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M85.fvp_mps2.sse200.crypto_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M85.fvp_mps2.sse200.dbg_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M85.fvp_mps2.sse200.exclusive_monitor_iotss_internal_sram0**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M85.fvp_mps2.sse200.exclusive_monitor_iotss_internal_sram1**
Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_Cortex_M85.fvp_mps2.sse200.exclusive_monitor_iotss_internal_sram2**
Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_Cortex_M85.fvp_mps2.sse200.exclusive_monitor_iotss_internal_sram3**
Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_Cortex_M85.fvp_mps2.sse200.exclusive_squasher**
Squashes the exclusive attribute on bus transactions.
Type: `PVBusExclusiveSquasher`.

**FVP_MPS2_Cortex_M85.fvp_mps2.sse200.idau_labeller**
Type: `LabellerIdauSecurity`.

**FVP_MPS2_Cortex_M85.fvp_mps2.sse200.internal_msniper_tcm**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M85.fvp_mps2.sse200.iotss_cpuidentity**
IoT Subsystem CPU_IDENTITY registers.
Type: `IoTSS_CPUIdentity`.

**FVP_MPS2_Cortex_M85.fvp_mps2.sse200.iotss_internal_sram0**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M85.fvp_mps2.sse200.iotss_internal_sram1**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M85.fvp_mps2.sse200.iotss_internal_sram2**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M85.fvp_mps2.sse200.iotss_internal_sram3**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M85.fvp_mps2.sse200.iotss_systemcontrol**
IoT Subsystem System Control registers.
Type: `IoTSS_SystemControl`.

**FVP_MPS2_Cortex_M85.fvp_mps2.sse200.iotss_systeminfo**
IoT Subsystem System Information registers.
Type: `IoTSS_SystemInfo`.

**FVP_MPS2_Cortex_M85.fvp_mps2.sse200.mem_switch_internal_msniper_tcm**
Allow transactions to be routed arbitrarily.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: PVBusRouter.

**FVP_MPS2_Cortex_M85.fvp_mps2.sse200.mem_switch_internal_sram**

Allow transactions to be routed arbitrarily.
Type: PVBusRouter.

**FVP_MPS2_Cortex_M85.fvp_mps2.sse200.mem_switch_internal_sram_mpc**

Allow transactions to be routed arbitrarily.
Type: PVBusRouter.

**FVP_MPS2_Cortex_M85.fvp_mps2.sse200.mem_switch_ppu**

Allow transactions to be routed arbitrarily.
Type: PVBusRouter.

**FVP_MPS2_Cortex_M85.fvp_mps2.sse200.mhu0**

IoT Subsystem Message Handling Unit.
Type: IoTSS_MessageHandlingUnit.

**FVP_MPS2_Cortex_M85.fvp_mps2.sse200.mhu1**

IoT Subsystem Message Handling Unit.
Type: IoTSS_MessageHandlingUnit.

**FVP_MPS2_Cortex_M85.fvp_mps2.sse200.mpc_iotss_internal_sram0**

IoT Subsystem Memory Protection Controller.
Type: IoTSS_MemoryProtectionController.

**FVP_MPS2_Cortex_M85.fvp_mps2.sse200.mpc_iotss_internal_sram0.gating_disabled_thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: SchedulerThreadEvent.

**FVP_MPS2_Cortex_M85.fvp_mps2.sse200.mpc_iotss_internal_sram1**

IoT Subsystem Memory Protection Controller.
Type: IoTSS_MemoryProtectionController.

**FVP_MPS2_Cortex_M85.fvp_mps2.sse200.mpc_iotss_internal_sram1.gating_disabled_thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: SchedulerThreadEvent.

**FVP_MPS2_Cortex_M85.fvp_mps2.sse200.mpc_iotss_internal_sram2**

IoT Subsystem Memory Protection Controller.
Type: IoTSS_MemoryProtectionController.

**FVP_MPS2_Cortex_M85.fvp_mps2.sse200.mpc_iotss_internal_sram2.gating_disabled_thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: SchedulerThreadEvent.

**FVP_MPS2_Cortex_M85.fvp_mps2.sse200.mpc_iotss_internal_sram3**

IoT Subsystem Memory Protection Controller.
Type: IoTSS_MemoryProtectionController.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M85.fvp_mps2.sse200.mpc_iotss_internal_sram3.gating_disabled_thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_Cortex_M85.fvp_mps2.sse200.nmi_or_gate**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_Cortex_M85.fvp_mps2.sse200.nonsecure_watchdog**

ARM Watchdog Module.
Type: `CMSDK_Watchdog`.

**FVP_MPS2_Cortex_M85.fvp_mps2.sse200.ram0_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M85.fvp_mps2.sse200.ram1_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M85.fvp_mps2.sse200.ram2_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M85.fvp_mps2.sse200.ram3_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M85.fvp_mps2.sse200.s32k_timer**

ARM Timer Module.
Type: `CMSDK_Timer`.

**FVP_MPS2_Cortex_M85.fvp_mps2.sse200.s32k_timer.clk_div**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M85.fvp_mps2.sse200.s32k_timer.counter**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M85.fvp_mps2.sse200.s32k_watchdog**

ARM Watchdog Module.
Type: `CMSDK_Watchdog`.

**FVP_MPS2_Cortex_M85.fvp_mps2.sse200.secure_control_register_block**

MPS2 Secure Control Register Block.
Type: `MPS2_SecureCtrl`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M85.fvp_mps2.sse200.secure_watchdog**

ARM Watchdog Module.
Type: `CMSDK_Watchdog`.

**FVP_MPS2_Cortex_M85.fvp_mps2.sse200.signal_router**

Signal router.
Type: `SignalRouter`.

**FVP_MPS2_Cortex_M85.fvp_mps2.sse200.sys_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_Cortex_M85.fvp_mps2.sse200.timer0**

ARM Timer Module.
Type: `CMSDK_Timer`.

**FVP_MPS2_Cortex_M85.fvp_mps2.sse200.timer0.clk_div**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M85.fvp_mps2.sse200.timer0.counter**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M85.fvp_mps2.sse200.timer1**

ARM Timer Module.
Type: `CMSDK_Timer`.

**FVP_MPS2_Cortex_M85.fvp_mps2.sse200.timer1.clk_div**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M85.fvp_mps2.sse200.timer1.counter**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M85.fvp_mps2.ssram1**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M85.fvp_mps2.ssram2**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M85.fvp_mps2.stub0**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M85.fvp_mps2.stub1**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M85.fvp_mps2.stub_i2c1**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M85.fvp_mps2.stub_i2s**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M85.fvp_mps2.stub_spi0**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M85.fvp_mps2.stub_spi2**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_Cortex_M85.fvp_mps2.svos_dualtimer**

Type: `SVOS_DualTimer`.

**FVP_MPS2_Cortex_M85.fvp_mps2.svos_dualtimer.svos_dualtimer0**

ARM Dual-Timer Module.
Type: `CMSDK_DualTimer`.

**FVP_MPS2_Cortex_M85.fvp_mps2.svos_dualtimer.svos_dualtimer0.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M85.fvp_mps2.svos_dualtimer.svos_dualtimer0.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M85.fvp_mps2.svos_dualtimer.svos_dualtimer0.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M85.fvp_mps2.svos_dualtimer.svos_dualtimer0.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M85.fvp_mps2.svos_dualtimer.svos_dualtimer1**

ARM Dual-Timer Module.
Type: `CMSDK_DualTimer`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M85.fvp_mps2.svos_dualtimer.svos_dualtimer1.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M85.fvp_mps2.svos_dualtimer.svos_dualtimer1.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M85.fvp_mps2.svos_dualtimer.svos_dualtimer1.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M85.fvp_mps2.svos_dualtimer.svos_dualtimer1.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M85.fvp_mps2.svos_dualtimer.svos_dualtimer2**

ARM Dual-Timer Module.
Type: `CMSDK_DualTimer`.

**FVP_MPS2_Cortex_M85.fvp_mps2.svos_dualtimer.svos_dualtimer2.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M85.fvp_mps2.svos_dualtimer.svos_dualtimer2.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M85.fvp_mps2.svos_dualtimer.svos_dualtimer2.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M85.fvp_mps2.svos_dualtimer.svos_dualtimer2.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M85.fvp_mps2.svos_dualtimer.svos_dualtimer3**

ARM Dual-Timer Module.
Type: `CMSDK_DualTimer`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_Cortex_M85.fvp_mps2.svos_dualtimer.svos_dualtimer3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M85.fvp_mps2.svos_dualtimer.svos_dualtimer3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_Cortex_M85.fvp_mps2.svos_dualtimer.svos_dualtimer3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M85.fvp_mps2.svos_dualtimer.svos_dualtimer3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_Cortex_M85.fvp_mps2.switch_PSRAM_M7**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M85.fvp_mps2.switch_svos_dualtimer**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_Cortex_M85.fvp_mps2.telnetterminal0**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_MPS2_Cortex_M85.fvp_mps2.telnetterminal1**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_MPS2_Cortex_M85.fvp_mps2.telnetterminal2**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_MPS2_Cortex_M85.fvp_mps2.touchscreen_interface**

MPS2 Touch Screen.
Type: `MPS2_TouchScreen`.

**FVP_MPS2_Cortex_M85.fvp_mps2.uart_overflows_or_gate**

Or Gate.
Type: `OrGate`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

# 16.12 FVP_MPS2_SSE-200_Cortex-M33

List of instances in FVP_MPS2_SSE-200_Cortex-M33.

## FVP_MPS2_SSE-200_Cortex-M33 instances

**FVP_MPS2_SSE_200_Cortex_M33**

Type: `FVP_MPS2_SSE_200_Cortex_M33`.

**FVP_MPS2_SSE_200_Cortex_M33.clk25Mhz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_SSE_200_Cortex_M33.clk25khz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_SSE_200_Cortex_M33.cpu0**

ARM CORTEXM33 CT model.
Type: `ARM_Cortex-M33`.

**FVP_MPS2_SSE_200_Cortex_M33.cpu1**

ARM CORTEXM33 CT model.
Type: `ARM_Cortex-M33`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2**

MPS2 DUT.
Type: `FVP_MPS2`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.GPIO0**

ARM PrimeCell General Purpose Input/Output.
Type: `CMSDK_GPIO`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.GPIO1**

ARM PrimeCell General Purpose Input/Output.
Type: `CMSDK_GPIO`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.GPIO2**

ARM PrimeCell General Purpose Input/Output.
Type: `CMSDK_GPIO`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.GPIO3**

ARM PrimeCell General Purpose Input/Output.
Type: `CMSDK_GPIO`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.GPIO_connection_test**

Type: `GPIO_Connection_Test`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.GPIO_connection_test.GPIO0_port_trans**

Type: `GPIO_Port_Transfer`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.GPIO_connection_test.GPIO1_port_test**

Type: `GPIO1_Connection_Test`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.PSRAM**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.PSRAM_M7**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.UART0**

ARM CMSDK UART Module.
Type: `CMSDK_UART`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.UART0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.UART1**

ARM CMSDK UART Module.
Type: `CMSDK_UART`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.UART1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.UART2**

ARM CMSDK UART Module.
Type: `CMSDK_UART`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.UART2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.VGA_interface**

VGA display interface between main bus and visualisation.
Type: `MPS2_VGA`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.ahb_ppc_iotss_expansion0**
IoT Subsystem Peripheral Protection Controller.
Type: `IoTSS_PeripheralProtectionController`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.ahb_ppc_iotss_expansion1**
IoT Subsystem Peripheral Protection Controller.
Type: `IoTSS_PeripheralProtectionController`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.apb_ppc_iotss_expansion0**
IoT Subsystem Peripheral Protection Controller.
Type: `IoTSS_PeripheralProtectionController`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.apb_ppc_iotss_expansion1**
IoT Subsystem Peripheral Protection Controller.
Type: `IoTSS_PeripheralProtectionController`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.apb_ppc_iotss_expansion2**
IoT Subsystem Peripheral Protection Controller.
Type: `IoTSS_PeripheralProtectionController`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.clock50Hz**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.clockdivider**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.cmsdk_sysctrl**
Cortex-M Simple System Control.
Type: `CMSDK_SysCtrl`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.cmsdk_watchdog**
ARM Watchdog Module.
Type: `CMSDK_Watchdog`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.cpu_wait_or_gate_0**
Or Gate.
Type: `OrGate`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.cpu_wait_or_gate_1**
Or Gate.
Type: `OrGate`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.dbgen_or_gate**
Or Gate.
Type: `OrGate`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.dma0**

ARM PrimeCell DMA Controller(PL080/081).
Type: `PL080_DMAC`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.dma0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.dma0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.dma0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.dma0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.dma0_idau_labeller**

Type: `LabellerIdauSecurity`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.dma0_securitymodifier**

Type: `SecurityModifier`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.dma1**

ARM PrimeCell DMA Controller(PL080/081).
Type: `PL080_DMAC`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.dma1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.dma1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.dma1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.dma1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.dma1_idau_labeller**

Type: `LabellerIdauSecurity`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.dma1_securitymodifier**

Type: `SecurityModifier`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.dma2**

ARM PrimeCell DMA Controller(PL080/081).
Type: `PL080_DMAC`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.dma2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.dma2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.dma2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.dma2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.dma2_idau_labeller**

Type: `LabellerIdauSecurity`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.dma2_securitymodifier**

Type: `SecurityModifier`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.dma3**

ARM PrimeCell DMA Controller(PL080/081).
Type: `PL080_DMAC`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.dma3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.dma3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.dma3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.dma3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.dma3_idau_labeller**

Type: `LabellerIdauSecurity`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.dma3_securitymodifier**

Type: `SecurityModifier`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.exclusive_monitor_psram**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.exclusive_monitor_psram_iotss**
Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.exclusive_monitor_switch_extra_psram_iotss**
Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.exclusive_monitor_switch_extra_psram_mps2**
Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.exclusive_monitor_zbtsram1**
Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.exclusive_monitor_zbtsram2**
Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.exclusive_squasher**
Squashes the exclusive attribute on bus transactions.
Type: `PVBusExclusiveSquasher`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.fpga_sysctrl**
FPGA SysCtrl timers LEDs and switches.
Type: `FPGA_SysCtrl`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.fpga_sysctrl.callBack100HzCounter**
Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.fpga_sysctrl.clockdivider**
A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.gpio_0_or_2**
Or Gate.
Type: `OrGate`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.gpio_1_or_3**
Or Gate.
Type: `OrGate`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.hostbridge**
Host Socket Interface Component.
Type: `HostBridge`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.mem_switch_extra_psram_iotss**
Allow transactions to be routed arbitrarily.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `PVBusRouter`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.mem_switch_extra_psram_mps2**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.mpc_iotss_ssram1**

IoT Subsystem Memory Protection Controller.
Type: `IoTSS_MemoryProtectionController`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.mpc_iotss_ssram1.gating_disabled_thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.mpc_iotss_ssram2**

IoT Subsystem Memory Protection Controller.
Type: `IoTSS_MemoryProtectionController`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.mpc_iotss_ssram2.gating_disabled_thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.mpc_iotss_ssram3**

IoT Subsystem Memory Protection Controller.
Type: `IoTSS_MemoryProtectionController`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.mpc_iotss_ssram3.gating_disabled_thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.mps2_audio**

MPS2 Audio.
Type: `MPS2_Audio`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.mps2_cmsdk_dualtimer**

ARM Dual-Timer Module.
Type: `CMSDK_DualTimer`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.mps2_cmsdk_dualtimer.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.mps2_cmsdk_dualtimer.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `ClockDivider`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.mps2_cmsdk_dualtimer.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.mps2_cmsdk_dualtimer.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.mps2_exclusive_monitor_zbtsram1**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.mps2_exclusive_monitor_zbtsram2**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.mps2_lcd**

MPS2 LCD I2C interface.
Type: `MPS2_LCD`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.mps2_mpc_iotss_ssram1**

IoT Subsystem Memory Protection Controller.
Type: `IoTSS_MemoryProtectionController`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.mps2_mpc_iotss_ssram1.gating_disabled_thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.mps2_secure_control_register_block**

MPS2 Secure Control Register Block.
Type: `MPS2_SecureCtrl`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.mps2_timer0**

ARM Timer Module.
Type: `CMSDK_Timer`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.mps2_timer0.clk_div**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.mps2_timer0.counter**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.mps2_timer1**

ARM Timer Module.
Type: `CMSDK_Timer`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.mps2_timer1.clk_div**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.mps2_timer1.counter**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.mps2_visualisation**

Display window for MPS2 using Visualisation library.
Type: `MPS2_Visualisation`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.niden_or_gate**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.nmi_or_gate**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.pl022_ssp_mps2**

ARM PrimeCell Synchronous Serial Port(PL022).
Type: `PL022_SSP_MPS2`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.pl022_ssp_mps2.prescaler**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.platform_bus_switch**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.platform_switch_dma0**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.platform_switch_dma1**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.signal_router**

Signal router.
Type: `SignalRouter`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.smsc_91c111**

SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.spiden_or_gate**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.spniden_or_gate**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.sse200**

SSE-200 subsystem.
Type: `SSE200`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.sse200.acg_cpu0**

IoT Subsystem Access Control Gate.
Type: `IoTSS_AccessControlGate`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.sse200.acg_cpu1**

IoT Subsystem Access Control Gate.
Type: `IoTSS_AccessControlGate`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.sse200.acg_sram0**

IoT Subsystem Access Control Gate.
Type: `IoTSS_AccessControlGate`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.sse200.acg_sram1**

IoT Subsystem Access Control Gate.
Type: `IoTSS_AccessControlGate`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.sse200.acg_sram2**

IoT Subsystem Access Control Gate.
Type: `IoTSS_AccessControlGate`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.sse200.acg_sram3**

IoT Subsystem Access Control Gate.
Type: `IoTSS_AccessControlGate`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.sse200.apb_ppc_iotss_subsystem0**

IoT Subsystem Peripheral Protection Controller.
Type: `IoTSS_PeripheralProtectionController`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.sse200.apb_ppc_iotss_subsystem1**

IoT Subsystem Peripheral Protection Controller.
Type: `IoTSS_PeripheralProtectionController`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.sse200.clock32kHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.sse200.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.sse200.cmsdk_dualtimer**

ARM Dual-Timer Module.
Type: `CMSDK_DualTimer`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.sse200.cmsdk_dualtimer.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.sse200.cmsdk_dualtimer.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.sse200.cmsdk_dualtimer.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.sse200.cmsdk_dualtimer.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.sse200.cordio_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.sse200.cpu0core_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.sse200.cpu0dbg_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.sse200.cpu1core_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.sse200.cpu1dbg_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.sse200.crypto_ppu**
ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.sse200.dbg_ppu**
ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.sse200.exclusive_monitor_iotss_internal_sram0**
Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.sse200.exclusive_monitor_iotss_internal_sram1**
Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.sse200.exclusive_monitor_iotss_internal_sram2**
Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.sse200.exclusive_monitor_iotss_internal_sram3**
Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.sse200.exclusive_squasher**
Squashes the exclusive attribute on bus transactions.
Type: `PVBusExclusiveSquasher`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.sse200.idau_labeller**
Type: `LabellerIdauSecurity`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.sse200.internal_msniper_tcm**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.sse200.iotss_cpuidentity**
IoT Subsystem CPU_IDENTITY registers.
Type: `IoTSS_CPUIdentity`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.sse200.iotss_internal_sram0**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.sse200.iotss_internal_sram1**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.sse200.iotss_internal_sram2**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.sse200.iotss_internal_sram3**
RAM device, can be dynamic or static ram.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `RAMDevice`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.sse200.iotss_systemcontrol**

IoT Subsystem System Control registers.
Type: `IoTSS_SystemControl`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.sse200.iotss_systeminfo**

IoT Subsystem System Information registers.
Type: `IoTSS_SystemInfo`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.sse200.mem_switch_internal_msniper_tcm**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.sse200.mem_switch_internal_sram**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.sse200.mem_switch_internal_sram_mpc**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.sse200.mem_switch_ppu**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.sse200.mhu0**

IoT Subsystem Message Handling Unit.
Type: `IoTSS_MessageHandlingUnit`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.sse200.mhu1**

IoT Subsystem Message Handling Unit.
Type: `IoTSS_MessageHandlingUnit`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.sse200.mpc_iotss_internal_sram0**

IoT Subsystem Memory Protection Controller.
Type: `IoTSS_MemoryProtectionController`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.sse200.mpc_iotss_internal_sram0.gating_disabled_thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.sse200.mpc_iotss_internal_sram1**

IoT Subsystem Memory Protection Controller.
Type: `IoTSS_MemoryProtectionController`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.sse200.mpc_iotss_internal_sram1.gating_disabled_thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.sse200.mpc_iotss_internal_sram2**

IoT Subsystem Memory Protection Controller.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `IoTSS_MemoryProtectionController`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.sse200.mpc_iotss_internal_sram2.gating_disabled_thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.sse200.mpc_iotss_internal_sram3**

IoT Subsystem Memory Protection Controller.
Type: `IoTSS_MemoryProtectionController`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.sse200.mpc_iotss_internal_sram3.gating_disabled_thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.sse200.nmi_or_gate**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.sse200.nonsecure_watchdog**

ARM Watchdog Module.
Type: `CMSDK_Watchdog`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.sse200.ram0_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.sse200.ram1_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.sse200.ram2_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.sse200.ram3_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.sse200.s32k_timer**

ARM Timer Module.
Type: `CMSDK_Timer`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.sse200.s32k_timer.clk_div**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.sse200.s32k_timer.counter**

Internal component used by SP804 Timer module.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `CounterModule`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.sse200.s32k_watchdog**

ARM Watchdog Module.
Type: `CMSDK_Watchdog`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.sse200.secure_control_register_block**

MPS2 Secure Control Register Block.
Type: `MPS2_SecureCtrl`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.sse200.secure_watchdog**

ARM Watchdog Module.
Type: `CMSDK_Watchdog`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.sse200.signal_router**

Signal router.
Type: `SignalRouter`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.sse200.sys_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.sse200.timer0**

ARM Timer Module.
Type: `CMSDK_Timer`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.sse200.timer0.clk_div**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.sse200.timer0.counter**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.sse200.timer1**

ARM Timer Module.
Type: `CMSDK_Timer`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.sse200.timer1.clk_div**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.sse200.timer1.counter**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.ssram1**

RAM device, can be dynamic or static ram.
Type: RAMDevice.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.ssram2**

RAM device, can be dynamic or static ram.
Type: RAMDevice.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.stub0**

RAM device, can be dynamic or static ram.
Type: RAMDevice.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.stub1**

RAM device, can be dynamic or static ram.
Type: RAMDevice.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.stub_i2c1**

RAM device, can be dynamic or static ram.
Type: RAMDevice.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.stub_i2s**

RAM device, can be dynamic or static ram.
Type: RAMDevice.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.stub_spi0**

RAM device, can be dynamic or static ram.
Type: RAMDevice.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.stub_spi2**

RAM device, can be dynamic or static ram.
Type: RAMDevice.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.svos_dualtimer**

Type: SVOS_DualTimer.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.svos_dualtimer.svos_dualtimer0**

ARM Dual-Timer Module.
Type: CMSDK_DualTimer.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.svos_dualtimer.svos_dualtimer0.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: ClockDivider.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.svos_dualtimer.svos_dualtimer0.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: ClockDivider.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.svos_dualtimer.svos_dualtimer0.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.svos_dualtimer.svos_dualtimer0.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.svos_dualtimer.svos_dualtimer1**

ARM Dual-Timer Module.
Type: `CMSDK_DualTimer`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.svos_dualtimer.svos_dualtimer1.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.svos_dualtimer.svos_dualtimer1.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.svos_dualtimer.svos_dualtimer1.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.svos_dualtimer.svos_dualtimer1.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.svos_dualtimer.svos_dualtimer2**

ARM Dual-Timer Module.
Type: `CMSDK_DualTimer`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.svos_dualtimer.svos_dualtimer2.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.svos_dualtimer.svos_dualtimer2.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.svos_dualtimer.svos_dualtimer2.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.svos_dualtimer.svos_dualtimer2.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.svos_dualtimer.svos_dualtimer3**

ARM Dual-Timer Module.
Type: `CMSDK_DualTimer`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.svos_dualtimer.svos_dualtimer3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.svos_dualtimer.svos_dualtimer3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.svos_dualtimer.svos_dualtimer3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.svos_dualtimer.svos_dualtimer3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.switch_PSRAM_M7**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.switch_svos_dualtimer**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.telnetterminal0**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.telnetterminal1**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.telnetterminal2**

Telnet terminal interface.
Type: `TelnetTerminal`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.touchscreen_interface**

MPS2 Touch Screen.
Type: `MPS2_TouchScreen`.

**FVP_MPS2_SSE_200_Cortex_M33.fvp_mps2.uart_overflows_or_gate**

Or Gate.
Type: `OrGate`.

## 16.13 FVP_MPS2_SSE-200_Cortex-M55

List of instances in FVP_MPS2_SSE-200_Cortex-M55.

### FVP_MPS2_SSE-200_Cortex-M55 instances

**FVP_MPS2_SSE_200_Cortex_M55**

Type: `FVP_MPS2_SSE_200_Cortex_M55`.

**FVP_MPS2_SSE_200_Cortex_M55.clk25Mhz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_SSE_200_Cortex_M55.clk25khz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_SSE_200_Cortex_M55.cpu0**

ARM Cortex-M55 CT model.
Type: `ARM_Cortex-M55`.

**FVP_MPS2_SSE_200_Cortex_M55.cpu1**

ARM Cortex-M55 CT model.
Type: `ARM_Cortex-M55`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2**

MPS2 DUT.
Type: `FVP_MPS2`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.GPIO0**

ARM PrimeCell General Purpose Input/Output.
Type: `CMSDK_GPIO`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.GPIO1**

ARM PrimeCell General Purpose Input/Output.
Type: `CMSDK_GPIO`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.GPIO2**

ARM PrimeCell General Purpose Input/Output.
Type: `CMSDK_GPIO`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.GPIO3**

ARM PrimeCell General Purpose Input/Output.
Type: `CMSDK_GPIO`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.GPIO_connection_test**

Type: `GPIO_Connection_Test`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.GPIO_connection_test.GPIO0_port_trans**

Type: `GPIO_Port_Transfer`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.GPIO_connection_test.GPIO1_port_test**

Type: `GPIO1_Connection_Test`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.PSRAM**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.PSRAM_M7**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.UART0**

ARM CMSDK UART Module.
Type: `CMSDK_UART`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.UART0.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.UART1**

ARM CMSDK UART Module.
Type: `CMSDK_UART`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.UART1.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.UART2**

ARM CMSDK UART Module.
Type: `CMSDK_UART`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.UART2.clk_divider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.VGA_interface**

VGA display interface between main bus and visualisation.
Type: `MPS2_VGA`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.ahb_ppc_iotss_expansion0**

IoT Subsystem Peripheral Protection Controller.
Type: `IoTSS_PeripheralProtectionController`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.ahb_ppc_iotss_expansion1**

IoT Subsystem Peripheral Protection Controller.
Type: `IoTSS_PeripheralProtectionController`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.apb_ppc_iotss_expansion0**

IoT Subsystem Peripheral Protection Controller.
Type: `IoTSS_PeripheralProtectionController`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.apb_ppc_iotss_expansion1**

IoT Subsystem Peripheral Protection Controller.
Type: `IoTSS_PeripheralProtectionController`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.apb_ppc_iotss_expansion2**

IoT Subsystem Peripheral Protection Controller.
Type: `IoTSS_PeripheralProtectionController`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.clock50Hz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.cmsdk_sysctrl**

Cortex-M Simple System Control.
Type: `CMSDK_SysCtrl`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.cmsdk_watchdog**

ARM Watchdog Module.
Type: `CMSDK_Watchdog`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.cpu_wait_or_gate_0**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.cpu_wait_or_gate_1**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.dbgen_or_gate**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.dma0**

ARM PrimeCell DMA Controller(PL080/081).
Type: `PL080_DMAC`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.dma0.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.dma0.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.dma0.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.dma0.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.dma0_idau_labeller**

Type: `LabellerIdauSecurity`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.dma0_securitymodifier**

Type: `SecurityModifier`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.dma1**

ARM PrimeCell DMA Controller(PL080/081).

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `PL080_DMAC`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.dma1.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.dma1.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.dma1.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.dma1.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.dma1_idau_labeller**

Type: `LabellerIdauSecurity`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.dma1_securitymodifier**

Type: `SecurityModifier`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.dma2**

ARM PrimeCell DMA Controller(PL080/081).
Type: `PL080_DMAC`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.dma2.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.dma2.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.dma2.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.dma2.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.dma2_idau_labeller**

Type: `LabellerIdauSecurity`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.dma2_securitymodifier**

Type: `SecurityModifier`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.dma3**

ARM PrimeCell DMA Controller(PL080/081).
Type: `PL080_DMAC`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.dma3.timer**

A ClockTimerThread(64) is a drop-in replacement for a ClockTimer(64) component. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.dma3.timer.timer**

A ClockTimerThread64 is a drop-in replacement for ClockTimer64. The main difference to the ClockTimer component is that the ClockTimerThread runs the signal() callback from a proper scheduler thread. This mean that the signal() function may directly or indirectly invoke wait() functions to wait for time or events. This is not allowed for the ClockTimer component which does not use a thread. Components which issue bus transactions from within the timer signal() callback must use ClockTimerThread(64) rather than ClockTimer(64).
Type: `ClockTimerThread64`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.dma3.timer.timer.thread**

A SchedulerThread instance represents a co-routine thread in the simulation.
Type: `SchedulerThread`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.dma3.timer.timer.thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.dma3_idau_labeller**

Type: `LabellerIdauSecurity`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.dma3_securitymodifier**

Type: `SecurityModifier`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.exclusive_monitor_psram**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.exclusive_monitor_psram_iotss**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.exclusive_monitor_switch_extra_psram_iotss**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.exclusive_monitor_switch_extra_psram_mps2**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.exclusive_monitor_zbtsram1**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.exclusive_monitor_zbtsram2**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.exclusive_squasher**

Squashes the exclusive attribute on bus transactions.
Type: `PVBusExclusiveSquasher`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.fpga_sysctrl**

FPGA SysCtrl timers LEDs and switches.
Type: `FPGA_SysCtrl`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.fpga_sysctrl.callBack100HzCounter**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.fpga_sysctrl.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.gpio_0_or_2**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.gpio_1_or_3**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.hostbridge**

Host Socket Interface Component.
Type: `HostBridge`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.mem_switch_extra_psram_iotss**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.mem_switch_extra_psram_mps2**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.mpc_iotss_ssram1**

IoT Subsystem Memory Protection Controller.
Type: `IoTSS_MemoryProtectionController`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.mpc_iotss_ssram1.gating_disabled_thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.mpc_iotss_ssram2**

IoT Subsystem Memory Protection Controller.
Type: `IoTSS_MemoryProtectionController`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.mpc_iotss_ssram2.gating_disabled_thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.mpc_iotss_ssram3**

IoT Subsystem Memory Protection Controller.
Type: `IoTSS_MemoryProtectionController`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.mpc_iotss_ssram3.gating_disabled_thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.mps2_audio**

MPS2 Audio.
Type: `MPS2_Audio`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.mps2_cmsdk_dualtimer**

ARM Dual-Timer Module.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `CMSDK_DualTimer`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.mps2_cmsdk_dualtimer.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.mps2_cmsdk_dualtimer.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.mps2_cmsdk_dualtimer.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.mps2_cmsdk_dualtimer.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.mps2_exclusive_monitor_zbtsram1**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.mps2_exclusive_monitor_zbtsram2**

Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.mps2_lcd**

MPS2 LCD I2C interface.
Type: `MPS2_LCD`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.mps2_mpc_iotss_ssram1**

IoT Subsystem Memory Protection Controller.
Type: `IoTSS_MemoryProtectionController`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.mps2_mpc_iotss_ssram1.gating_disabled_thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.mps2_secure_control_register_block**

MPS2 Secure Control Register Block.
Type: `MPS2_SecureCtrl`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.mps2_timer0**

ARM Timer Module.
Type: `CMSDK_Timer`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.mps2_timer0.clk_div**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.mps2_timer0.counter**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.mps2_timer1**

ARM Timer Module.
Type: `CMSDK_Timer`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.mps2_timer1.clk_div**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.mps2_timer1.counter**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.mps2_visualisation**

Display window for MPS2 using Visualisation library.
Type: `MPS2_Visualisation`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.niden_or_gate**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.nmi_or_gate**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.pl022_ssp_mps2**

ARM PrimeCell Synchronous Serial Port(PL022).
Type: `PL022_SSP_MPS2`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.pl022_ssp_mps2.prescaler**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.platform_bus_switch**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.platform_switch_dma0**
Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.platform_switch_dma1**
Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.signal_router**
Signal router.
Type: `SignalRouter`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.smsc_91c111**
SMSC 91C111 ethernet controller.
Type: `SMSC_91C111`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.spiden_or_gate**
Or Gate.
Type: `OrGate`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.spniden_or_gate**
Or Gate.
Type: `OrGate`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.sse200**
SSE-200 subsystem.
Type: `SSE200`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.sse200.acg_cpu0**
IoT Subsystem Access Control Gate.
Type: `IoTSS_AccessControlGate`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.sse200.acg_cpu1**
IoT Subsystem Access Control Gate.
Type: `IoTSS_AccessControlGate`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.sse200.acg_sram0**
IoT Subsystem Access Control Gate.
Type: `IoTSS_AccessControlGate`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.sse200.acg_sram1**
IoT Subsystem Access Control Gate.
Type: `IoTSS_AccessControlGate`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.sse200.acg_sram2**
IoT Subsystem Access Control Gate.
Type: `IoTSS_AccessControlGate`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.sse200.acg_sram3**
IoT Subsystem Access Control Gate.
Type: `IoTSS_AccessControlGate`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.sse200.apb_ppc_iotss_subsystem0**

IoT Subsystem Peripheral Protection Controller.
Type: `IoTSS_PeripheralProtectionController`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.sse200.apb_ppc_iotss_subsystem1**

IoT Subsystem Peripheral Protection Controller.
Type: `IoTSS_PeripheralProtectionController`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.sse200.clock32kHz**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.sse200.clockdivider**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.sse200.cmsdk_dualtimer**

ARM Dual-Timer Module.
Type: `CMSDK_DualTimer`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.sse200.cmsdk_dualtimer.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.sse200.cmsdk_dualtimer.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.sse200.cmsdk_dualtimer.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.sse200.cmsdk_dualtimer.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.sse200.cordio_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.sse200.cpu0core_ppu**
ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.sse200.cpu0dbg_ppu**
ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.sse200.cpu1core_ppu**
ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.sse200.cpu1dbg_ppu**
ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.sse200.crypto_ppu**
ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.sse200.dbg_ppu**
ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.sse200.exclusive_monitor_iotss_internal_sram0**
Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.sse200.exclusive_monitor_iotss_internal_sram1**
Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.sse200.exclusive_monitor_iotss_internal_sram2**
Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.sse200.exclusive_monitor_iotss_internal_sram3**
Global exclusive monitor.
Type: `PVBusExclusiveMonitor`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.sse200.exclusive_squasher**
Squashes the exclusive attribute on bus transactions.
Type: `PVBusExclusiveSquasher`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.sse200.idau_labeller**
Type: `LabellerIdauSecurity`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.sse200.internal_msniper_tcm**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.sse200.iotss_cpuidentity**
IoT Subsystem CPU_IDENTITY registers.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Type: `IoTSS_CPUIdentity`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.sse200.iotss_internal_sram0**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.sse200.iotss_internal_sram1**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.sse200.iotss_internal_sram2**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.sse200.iotss_internal_sram3**
RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.sse200.iotss_systemcontrol**
IoT Subsystem System Control registers.
Type: `IoTSS_SystemControl`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.sse200.iotss_systeminfo**
IoT Subsystem System Information registers.
Type: `IoTSS_SystemInfo`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.sse200.mem_switch_internal_msniper_tcm**
Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.sse200.mem_switch_internal_sram**
Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.sse200.mem_switch_internal_sram_mpc**
Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.sse200.mem_switch_ppu**
Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.sse200.mhu0**
IoT Subsystem Message Handling Unit.
Type: `IoTSS_MessageHandlingUnit`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.sse200.mhu1**
IoT Subsystem Message Handling Unit.
Type: `IoTSS_MessageHandlingUnit`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.sse200.mpc_iotss_internal_sram0**
IoT Subsystem Memory Protection Controller.
Type: `IoTSS_MemoryProtectionController`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.sse200.mpc_iotss_internal_sram0.gating_disabled_thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.sse200.mpc_iotss_internal_sram1**

IoT Subsystem Memory Protection Controller.
Type: `IoTSS_MemoryProtectionController`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.sse200.mpc_iotss_internal_sram1.gating_disabled_thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.sse200.mpc_iotss_internal_sram2**

IoT Subsystem Memory Protection Controller.
Type: `IoTSS_MemoryProtectionController`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.sse200.mpc_iotss_internal_sram2.gating_disabled_thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.sse200.mpc_iotss_internal_sram3**

IoT Subsystem Memory Protection Controller.
Type: `IoTSS_MemoryProtectionController`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.sse200.mpc_iotss_internal_sram3.gating_disabled_thread_event**

A SchedulerThreadEvent instance is an auto-reset boolean condition variable other threads can wait on.
Type: `SchedulerThreadEvent`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.sse200.nmi_or_gate**

Or Gate.
Type: `OrGate`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.sse200.nonsecure_watchdog**

ARM Watchdog Module.
Type: `CMSDK_Watchdog`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.sse200.ram0_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.sse200.ram1_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.sse200.ram2_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.sse200.ram3_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.sse200.s32k_timer**

ARM Timer Module.
Type: `CMSDK_Timer`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.sse200.s32k_timer.clk_div**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.sse200.s32k_timer.counter**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.sse200.s32k_watchdog**

ARM Watchdog Module.
Type: `CMSDK_Watchdog`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.sse200.secure_control_register_block**

MPS2 Secure Control Register Block.
Type: `MPS2_SecureCtrl`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.sse200.secure_watchdog**

ARM Watchdog Module.
Type: `CMSDK_Watchdog`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.sse200.signal_router**

Signal router.
Type: `SignalRouter`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.sse200.sys_ppu**

ARM Power Policy Unit (PPU) architectural model.
Type: `PPUv0`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.sse200.timer0**

ARM Timer Module.
Type: `CMSDK_Timer`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.sse200.timer0.clk_div**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.sse200.timer0.counter**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.sse200.timer1**

ARM Timer Module.
Type: `CMSDK_Timer`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.sse200.timer1.clk_div**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.sse200.timer1.counter**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.ssram1**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.ssram2**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.stub0**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.stub1**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.stub_i2c1**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.stub_i2s**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.stub_spi0**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.stub_spi2**

RAM device, can be dynamic or static ram.
Type: `RAMDevice`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.svos_dualtimer**

Type: `SVOS_DualTimer`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.svos_dualtimer.svos_dualtimer0**

ARM Dual-Timer Module.
Type: `CMSDK_DualTimer`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.svos_dualtimer.svos_dualtimer0.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.svos_dualtimer.svos_dualtimer0.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.svos_dualtimer.svos_dualtimer0.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.svos_dualtimer.svos_dualtimer0.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.svos_dualtimer.svos_dualtimer1**

ARM Dual-Timer Module.
Type: `CMSDK_DualTimer`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.svos_dualtimer.svos_dualtimer1.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.svos_dualtimer.svos_dualtimer1.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.svos_dualtimer.svos_dualtimer1.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.svos_dualtimer.svos_dualtimer1.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.svos_dualtimer.svos_dualtimer2**

ARM Dual-Timer Module.
Type: `CMSDK_DualTimer`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.svos_dualtimer.svos_dualtimer2.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.svos_dualtimer.svos_dualtimer2.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.svos_dualtimer.svos_dualtimer2.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.svos_dualtimer.svos_dualtimer2.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.svos_dualtimer.svos_dualtimer3**

ARM Dual-Timer Module.
Type: `CMSDK_DualTimer`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.svos_dualtimer.svos_dualtimer3.clk_div0**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.svos_dualtimer.svos_dualtimer3.clk_div1**

A ClockDivider is a library component that takes a ClockSignal on its input port (which could come from the output of a MasterClock, or from another ClockDivider), and generates a new ClockSignal on its output port, representing a clock frequency that is related to the input clock by the ratio of the multiply and divide parameters.
Type: `ClockDivider`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.svos_dualtimer.svos_dualtimer3.counter0**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.svos_dualtimer.svos_dualtimer3.counter1**

Internal component used by SP804 Timer module.
Type: `CounterModule`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.switch_PSRAM_M7**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.switch_svos_dualtimer**

Allow transactions to be routed arbitrarily.
Type: `PVBusRouter`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.telnetterminal0**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.telnetterminal1**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.telnetterminal2**

Telnet terminal interface.
Type: `TelnetTerminal`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.touchscreen_interface**

MPS2 Touch Screen.
Type: `MPS2_TouchScreen`.

**FVP_MPS2_SSE_200_Cortex_M55.fvp_mps2.uart_overflows_or_gate**

Or Gate.
Type: `OrGate`.

Copyright © 2017–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential